COMMODORE

64

Felhasználói kézikönyv

NOVOTRADE

COMMODORE

64

Felhasználói kézikönyv

NOVOTRADE

A könyv eredeti címe: COMMODORE 64 Bedienungshandbuch

Fordította: Czifra András
Nagy Ákos
Lektorálta: Czinege Zoltán

Második, változatlan kiadás

A kiadásért felel Siba László, a Novotrade Kiadó Kft. ügyvezető igazgatója
Budapest, 1993

Szedés a Novotrade Rt. kiadójában készült

ISBN 963 585 188 X

Egyetemi Nyomda — 93.2513 Budapest, 1993
Felelős vezető: Sümeghi Zoltán igazgató

Tartalom

Bevezetés

Üdvözöljük az Olvasót a COMMODORE 64-es tulajdonosok népes táborában. Gépünk a személyi számítógép kategóriában az egyik legjobb a világon. Ugyanezt a színvonalat szeretnénk biztosítani a COMMODORE 64-es felhasználói kézikönyv esetében is, ezért nagy súlyt fektettünk az érthetőségre és a világos, logikus szerkezetre. Igyekeztünk minden információt megadni a gép biztonságos üzembe helyezéséhez és szakszerű kezeléséhez. A BASIC nyelv alapjainak ismertetésével a saját programok készítéséhez kívántunk segítséget nyújtani. Azok számára, akik megelégednek a készen kapható programok felhasználásával, az összes ehhez szükséges ismeretet a könyv elején sűrítettük össze.

Nézzük ezek után röviden, hogy mi mindenre is képes a COMMODORE 64-es! A grafika tekintetében bizonyára kategóriájának bajnoka. A konstruktőrök a SPRITE-grafika kitalálásával korábban szinte elképzelhetetlen animációs lehetőségeket biztosítottak. A képernyőn egyidejűleg 8 különböző általunk konstruált alakzatot mozgathattunk 3 színben anélkül, hogy ehhez bonyolult programozási trükkökre lenne szükségünk.

Az alakzatok "összeütközése" egyszerűen regisztrálható és előre programozott akciókkal köthető össze. Így váltható ki pl. egy robbanás a játékprogramokban két űrhajó ütközésekor.

A beépített hanggenerátor versenyre kelhet jó néhány szintetizátorral. Egyidejűleg három egymástól teljesen független szólamot szólaltathatunk meg, mindegyiket a teljes 8 oktáv hangterjedelemben. Négy hullámforma (négyszög, fűrészfog, háromszög, zaj) közül választhatunk. Rendelkezésünkre áll továbbá egy programozható burkológörbe generátor, felül-, alul-, ill. sávátmetsző szűrők, valamint egy hangerő- és rezonanciaszabályozó. A "komponált" zene megfelelő minőségű visszaadásához a COMMODORE 64-esre HiFi erősítőt köthetünk.

Ha már itt tartunk..., gépünk a különböző perifériás készülékek hozzákapcsolásával rendszerré bővíthető. A programok tárolásához egy kazettásegységet és max. 5 VC 1541-es típusú hajlékonylemezes meghajtót köthetünk össze a COMMODORE 64-essel. A VC mátrixnyomtatóval programlistáinkat, leveleinket stb. nyomtathatjuk ki. Ha érdeklődünk a CP/M operációs rendszer iránt, akkor gépünket egyszerűen elláthatjuk egy Z80-as mikroprocesszort tartalmazó kártyával.

Ez a felhasználói kézikönyv nem kevésbé fontos, mint az imént felsorolt ún. hardver elemek. Nem mond el mindent a számítógépről és a programozásról, de összefoglalja azokat az alapismereteket, amelyeknek a birtokában később elmélyedhetünk a rendelkezésre álló szakirodalomban. Ne felejtsük azonban el, hogy néhány nap alatt nem válhatunk gyakorlott programozóvá; legyünk tehát türelmesek és tanulmányozzuk át gondosan, pontról-pontra a kézikönyvet! Mindehhez sok sikert és kellemes szórakozást kívánunk.

1. FEJEZET

AZ ÜZEMBE HELYEZÉS

A COMMODORE 64-es üzembe helyezése

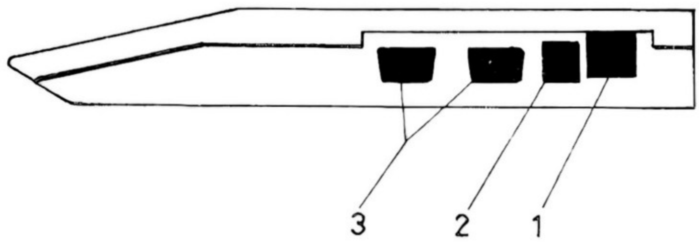
Ebből a fejezetből megtudhatjuk, hogyan kell összekapcsolni a gépet egy televíziókészülékkel, ill. egy monitorral. Először is vizsgáljuk meg a COMMODORE 64-es doboz tartalmát! Ennek a kézikönyvön kívül tartalmaznia kell

- a COMMODORE 64-es számítógépet
- a tápegységet és
- az antennakábelt.

Ha a felsoroltak közül bármelyik is hiányozna, kérjük forduljon az eladóhoz! Elsőként a csatlakozók elhelyezését és funkcióit tekintjük át.

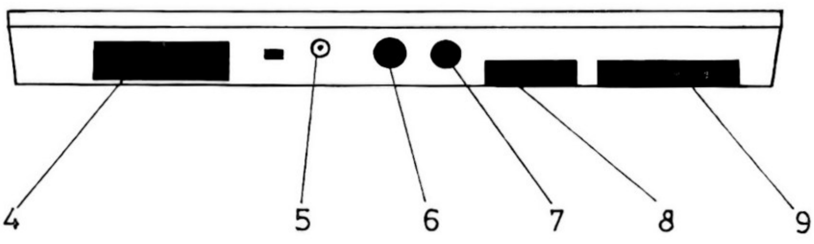
Oldalcsatlakozók

1. Hálózati csatlakozó. A gép energiaellátását biztosítja.
2. Be/kikapcsoló gomb.
3. Játékcsatlakozók. Botkormány (joystick), fényceruza és forgatógomb (paddle) illesztését teszik lehetővé.



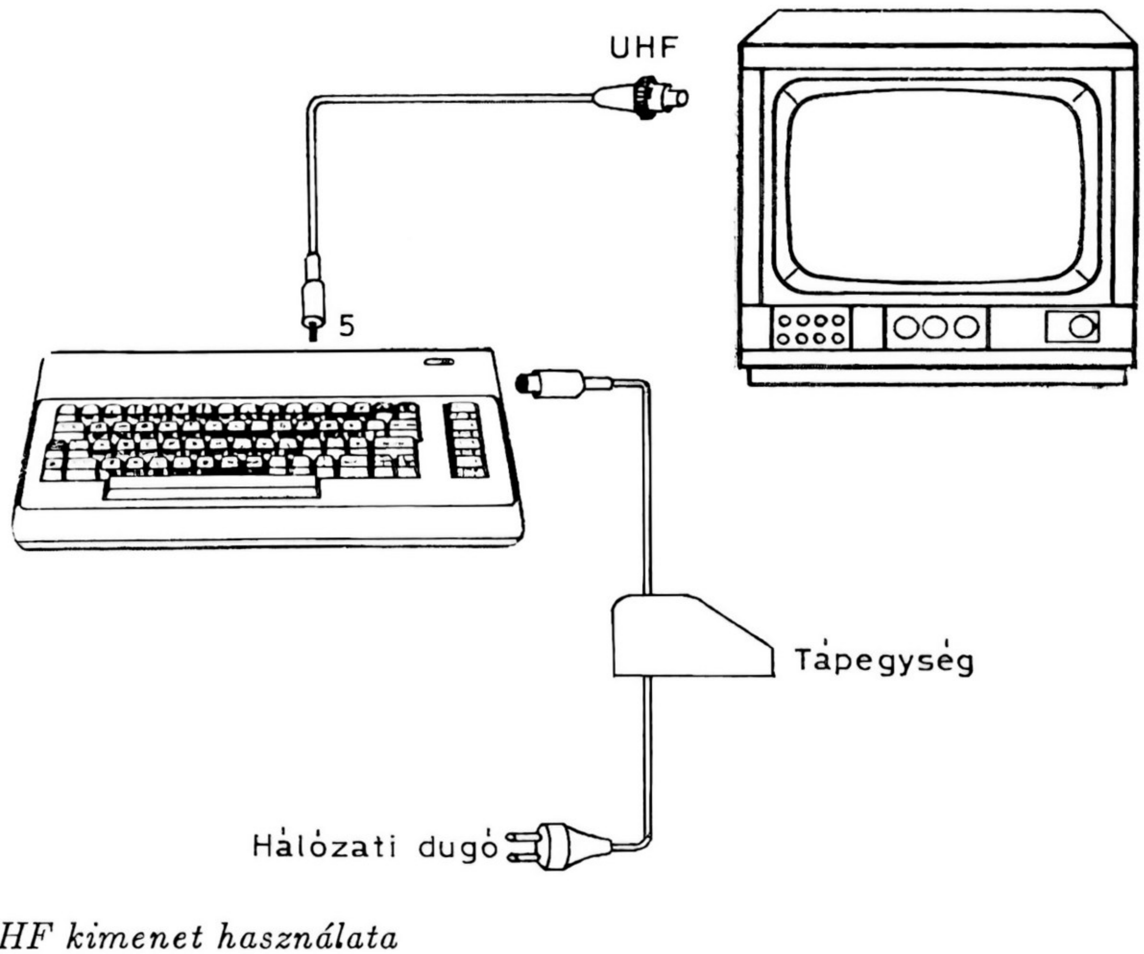
Hátlapcsatlakozók

4. Modulcsatlakozó program- és játékmodulok illesztéséhez.
5. Tv csatlakozó (75 Ω). A kimeneti jel a kép- és hanginformációt is tartalmazza.
6. Audio- és videokimenet. Itt férhetünk hozzá a közvetlen hangjelhez, amelyet pl. egy HI-FI berendezéshez vezethetünk. A videojellel egy monitort vezérelhetünk.
7. Soros vonali csatlakozó nyomtató és lemezmeghajtó illesztéséhez.
8. Magnetofoncsatlakozó a program- és adattároláshoz használt kazettásegység illesztéséhez.
9. Felhasználói modulok (pl. RS 232 vonal) illesztéséhez.



Összekapcsolás a tv-készülékkel

A csatlakoztatást az ábra alapján végezhetjük el.



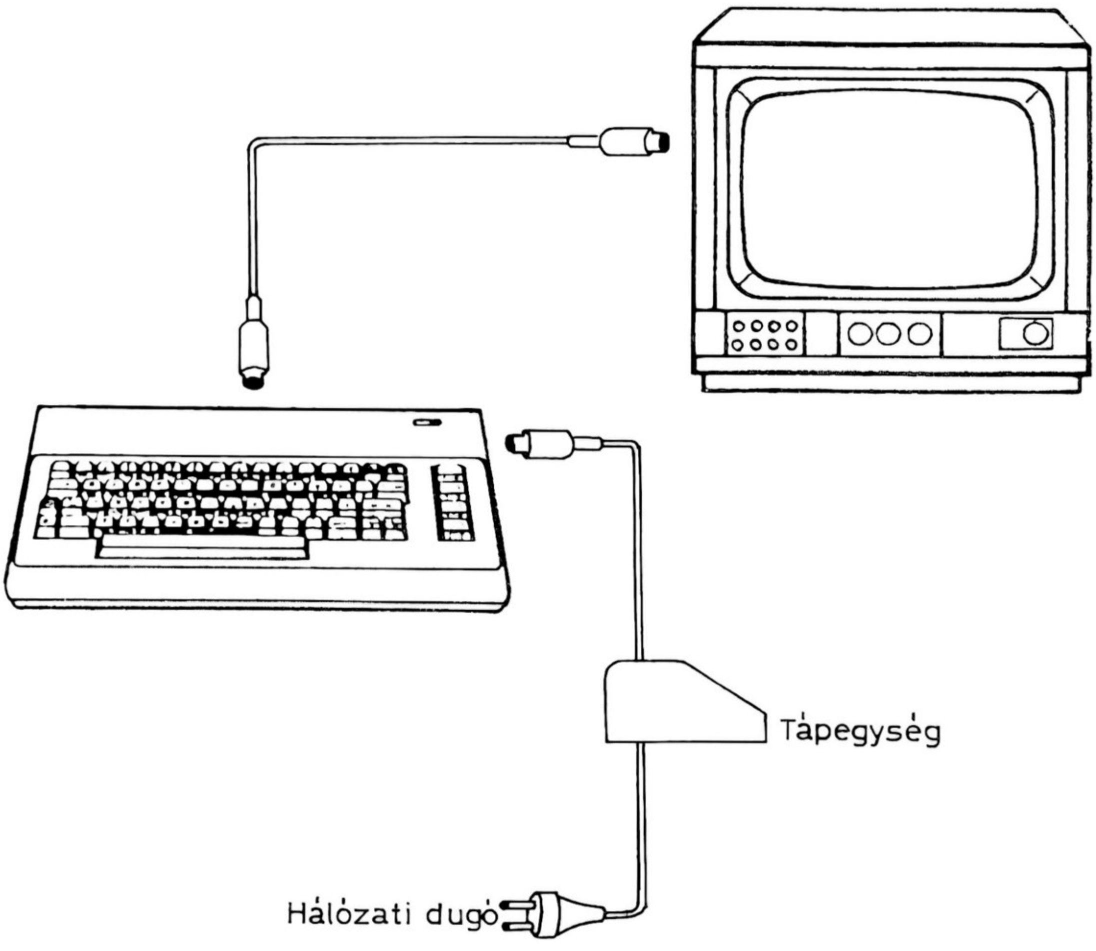
. *Az UHF kimenet használata*

A tv-kábelt használva kössük össze a COMMODORE 64-es tv-kimenetét (5) televíziónk UHF antenna bemenetével! A tv-készüléket a 36-os csatornára állítsuk!

. *A video-kimenet használata*

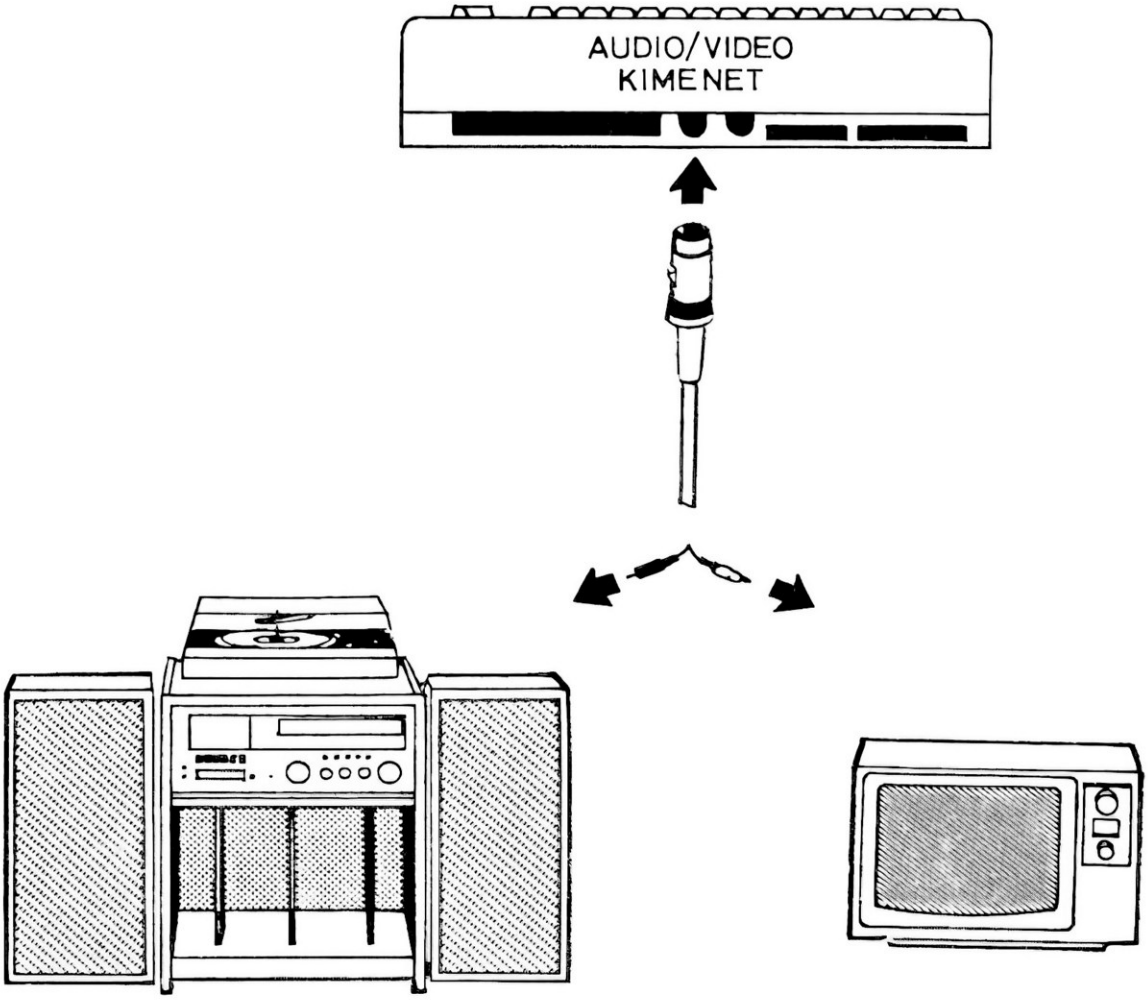
A legjobb minőségű képet videomonitorral kaphatjuk. Ekkor olyan videokábelt kell használnunk, amely 5 pólusú DIN csatlakozóval (DIN 41524) kapcsolódik gépünk 6-os videokimenetéhez. A kábel másik végét (DIN 45322) a monitorhoz csatlakoztatjuk. Ha tv-készülékünk videobemenettel is rendelkezik, akkor monitorként is használhatjuk, csupán arra kell ügyelnünk, hogy a készülék videocsatlakozója bemeneti állapotba legyen kapcsolva. Ha

videomagnetofont használunk az átkapcsolást rendszerint a csatlakozó 1-es lábára kapcsolt 12 V-os segédfeszültség végzi.
A COMMODORE 64-es azonban nem ad ki ilyen segédfeszültséget, ezért célszerű a tv-készülékbe szakemberrel egy átkapcsolót szereltetni. (Néhány televíziótípus a videocsatlakozó 5-ös lábán előállítja az átkapcsoláshoz szükséges 12 V-os segédfeszültséget.) A videobemenet használatakor gondosan ügyelnünk kell, nehogy a 12 V-os segédfeszültség számítógépünk kimeneteire kerüljön. Ez ugyanis elkerülhetetlenül tönkreteszi a gépet. Ezért jobb, ha az összekapcsolást szakemberre bízzuk.
A normál televízióadások vételekor a videokábelt rendszerint el kell távolítanunk.



3. *Hálózat*
   A tápegységet kapcsoljuk a gépre (1-es csatlakozó), ill. a másik oldalon a 220 V/50 Hz-es hálózatra!
4. *További csatlakozók.*
   A COMMODORE 64-es HiFi minőségű hangjelet állít elő, amelyet egy jó minőségű erősítővel felerősíthetünk. A hangjel a 6-os kimeneten érhető el. (A csatlakozó kiosztást lásd az I Függelékben!)

A géphez perifériaként hozzákapcsolhatunk továbbá pl. egy VC 1541-es hajlékonylemezes meghajtót, illetve egy VC 1525-ös nyomtatót. Ekkor a rendszer a következőképpen néz ki:



Bekapcsolás

. Kapcsoljuk be a gépet a jobb oldalán levő billenőkapcsolóval!

:. Néhány pillanat múlva a képernyőn a következőket látjuk:

```
****COMMODORE 64 BASIC V2****
64K RAM SYSTEM 38911 BASIC BYTES FREE

READY
```

Villogó kurzor
jelzi, hogy a gép
bevitelre vár.

. Ha a tv-készülékünkön van csatorna finombeállító gomb, hangoljuk a tv-t addig, amíg tiszta, éles képet nem kapunk! Ha a beállítás automatikus, akkor ezzel nem kell törődnünk.

4. A színek és a fényerő beállításával csak a következő szakaszban foglalkozunk. Most csak annyit kell elérnünk, hogy sötétkék képernyőt lássunk világoskék kerettel és világoskék betűkkel.

Ha a COMMODORE 64-es nem a leírtak szerint jelentkezik be a képernyőn, akkor vizsgáljuk meg újra rendszerünk összeállítását a következő táblázat alapján!

Hibakeresés

Jelenség	Ok	Javaslat
A gépen nem ég a piros jelzőlámpa	A gép nincs bekapcsolva	Győződjünk meg róla, hogy a gép jobb oldalán levő kapcsoló bekapcsolt állásban van-e!
	Nincs hálózati feszültség	Ellenőrizzük a tápegység hálózati és számítógép oldali csatlakoztatását!
	Hibás biztosíték	Cseréltessük ki a biztosítékot szakemberrel.
Nincs kép	A tv rossz csatornán van	Kapcsoljunk át a megfelelő csatornára!
	Hiányzó vagy téves antenna-, ill. videokábel csatlakoztatás.	Vizsgáljuk meg a kérdéses kábelt!
Zavaros kép egy modul bedugása után	A modul csatlakoztatása hibás	Kapcsoljuk ki a gépet, és vizsgáljuk meg a csatlakozó felületet, ill. a modul illeszkedését.
Gyenge színek	A tv-készülék helytelenül van beállítva	Ellenőrizzük a tv-készülék szín- és csatornabeállítását!

Jelenség	Ok	Javaslat
Erős háttérzaj	A tv-készülék hangerejét túl nagyra állítottuk	
A kép rendben van, de hang nincs	A tv-készülék hangerejét túl kicsire állítottuk	
	Az audio kimenet nincs összekötve az erősítő bemenetével	Kössük össze számítógépünk audiokimenetét az erősítő AUX bemenetével és a bemenetválasztó gombot állítsuk AUX-ra!

kurzor

A READY (parancs fogadására kész) felirat alatt villogó világos négyzetet ırzornak nevezzük. Azt a helyet jelzi a képernyőn, ahol a billentyűzeten leütött ırakter megjelenik. Gépeljünk be egy tetszőleges szöveget és figyeljük meg a kurzor ozgását a képernyőn!

A színek beállítása

A színbeállítást egy a képernyőn egyszerűen létrehozható minta segítségével égezhetjük el.

A minta előállításához a billentyűzet bal oldalán levő CTRL billentyűt asználjuk. (A név a ConTRoL (vezérlés) angol szó rövidítéséből származik.) Ezt a illentyűt mindig egy másikkal együtt kell lenyomnunk. Ilyenkor a számítógép egy ıeghatározott új működési módba kerül és rendszerint ebben is marad a következő tkapcsolásig.

Tartsuk lenyomva a CTRL billentyűt, majd üssük le a 9 gombot! Közvetlenül em tapasztalunk semmiféle változást. Ha azonban most leütünk egy billentyűt, kkor azt látjuk, hogy a megfelelő karakter inverz video módban jelenik meg a épernyőn (a háttér és a felirat színe felcserélődik).

Nyomjuk le most a SPACE (betűköz) billentyűt! Ha eddig mindent jól sináltunk, akkor a képernyőn egy világos csíkot látunk, amelynek hossza mindddig növekszik, amíg a billentyűt el nem engedjük.

A [CTRL] gomb és egy számbillentyű egyedejű lenyomása a rajzolószín megváltoztatását eredményezi. Kapcsoljunk most a [8] billentyűvel a sárga színre! Nyomjuk le újból a [SPACE] billentyűt!

Mi történik? Láthatjuk, hogy egy tetszőleges hosszúságú sárga csíkot rajzolhatunk.

A fenti módszerrel hozzunk létre egy színmintát, majd a tv-készülék szín- és fényerő-szabályozó gombjaival próbáljuk meg a lehető legerősebb, legtisztább színeket beállítani! (A megjelenő színeknek természetesen meg kell egyezniük a színbillentyűkön jelöltekkel.) A képernyő ekkor így néz ki:

```
****COMMODORE 64 BASIC V2****
64K RAM SYSTEM 38911 BASIC BYTES FREE

READY
```

3 Piros csík
6 Zöld csík
7 Kék csík
8 Sárga csík

A rendszer beállítását ezzel befejeztük.

A következő fejezetek bevezetnek a BASIC programozási nyelv használatába. Természetesen már kész programokat is kipróbálhatunk anélkül, hogy előzetes programozási ismeretekkel rendelkeznénk.

Az ilyen programcsomagok általában részletes kezelési útmutatást is tartalmaznak. Mégis célszerűnek tartjuk, hogy az újdonsült COMMODORE-tulajdonosok még ebben az esetben is áttanulmányozzák a kézikönyv első néhány fejezetét, hogy tisztában legyenek a gép alapvető sajátosságaival.

2. FEJEZET

ELSŐ LÉPÉSEK

A billentyűzet
Programok betöltése és tárolása
PRINT parancs és számolás a COMMODORE 64-essel
Számítási műveletek

A billentyűzet

Szenteljünk néhány percet a billentyűzettel való ismerkedésnek. Ez semmiképpen sem lesz elfecsérelt idő, mivel a billentyűzet az információ bevitelének legfontosabb eszköze.

A billentyűzet leginkább egy írógépre emlékeztet. Számos kiegészítő funkció van azonban; ezekkel most csak röviden foglalkozunk, a pontos leírást a későbbi fejezetek tartalmazzák.

RETURN (kocsivissza)

A RETURN billentyűvel arra utasítjuk a gépet, hogy a beírt, képernyőn látható karaktereket a tárba olvassa.

SHIFT (betűváltó)

A SHIFT billentyű nagyjából a szokványos írógépek betűváltójának felel meg. Segítségével a nagybetű/kisbetű üzemmódban a nagybetűket, a nagybetű/grafika üzemmódban pedig a billentyűk jobb oldalán jelölt grafikus szimbólumokat érhetjük el.

A többszörös funkcióbillentyűk (pl. CLR/HOME) esetében a felső részen levő (ebben az esetben CLR) funkciót választja ki.

Szerkesztés

Mindenki hibázhat, és ezt a COMMODORE 64-es figyelembe veszi. Számos funkcióbillentyű könnyíti meg a javítást.

CRSR (kurzor)

Talán még emlékszünk, hogy a képernyőn villogó négyzetet kurzornak neveztük. A CRSR billentyűkkel a kurzort a képernyőn a kívánt pontba vihetjük. A bal oldali CRSR billentyűvel a kurzort lefelé, illetve a SHIFT-tel együtt felfelé mozgathatjuk. A jobb oldali CRSR kurzorvezérlővel jobbra és (SHIFT-tel együtt) balra mozoghatunk. A kurzormozgató billentyűk ún. ismétlési funkcióval rendelkeznek, így a kurzor mindaddig mozog, amíg lenyomva tartjuk a billentyűt.

INST/DEL (beszúrás/törlés)

Ha egy szó begépelése után megnyomjuk az INST/DEL billentyűt, akkor

: utolsónak beírt karakter törlődik. Ha a kurzor a szó közepén áll, akkor az NST/DEL a kurzortól balra álló karaktert törli, a kurzortól jobbra levő szórész edig egy pozícióval balra lép. A SHIFT és INST/DEL billentyűk együttes nyomásával új karaktereket szúrhatunk be egy már meglevő sorba. Vigyük a urzort arra a helyre, ahonnan a kérdéses karakter hiányzik, majd nyomjuk le ;yidejűleg a SHIFT és INST/DEL billentyűket! Ekkor az a jel, amely felett kurzor áll és vele együtt a sornak a kurzortól jobbra eső része egy pozícióval bbra lép. Ekkor beírhatjuk a hiányzó karaktert.

:LR/HOME (törlés, alaphelyzet)

A CLR/HOME billentyű a kurzort a képernyő bal felső sarkába levő alap-elyzetbe (HOME (haza) pozíció) viszi. A SHIFT és a CLR/HOME együttes ütése törli a képernyőt, és a kurzor az alaphelyzetbe kerül.

ŁESTORE (visszaállítás)

A RESTORE és RUN/STOP billentyűk együttes lenyomása az ún. kiindulási lapotba állítja vissza a számítógépet. Hogy ez pontosan mit jelent, arról a ésőbbiekben adunk bővebb magyarázatot.

. funkcióbillentyűk

A billentyűzet jobb oldalán levő funkcióbillentyűket különféle tevékenységekre het "beprogramozni". Ilyenkor azt használjuk ki, hogy a négy billentyűnek a 133-l 136-ig (SHIFT-tel együtt 137-től 140-ig) terjedő ASCII kódok felelnek meg ásd az F Függeléket).

:TRL (vezérlés)

A CTRL billentyű a színek közötti átkapcsolásra, ill. az inverz video üzemmód e-, ill. kikapcsolására alkalmas. A CTRL billentyűt lenyomva kell tartanunk, nikor leütjük a megfelelő másik billentyűt.

ŁUN/STOP (futás/stop)

A RUN/STOP billentyű lenyomásával egy BASIC nyelvű program futtatását :akíthatjuk meg. A SHIFT-tel együtt egy program, kazettáról való tárba töltését : indítását eredményezi.

[C=] (COMMODORE billentyű)

Ennek a billentyűnek számos rendeltetése van. Segítségével érhetjük el pl. a billentyűk elülső oldalának bal felén ([SHIFT]-tel együtt a jobb felén) feltüntetett szimbólumokat.

Számítógépünk a bekapcsolás után automatikusan a nagybetű/grafika üzemmódba kerül. Ilyenkor a begépelt szöveg nagybetűvel jelenik meg, és valamennyi grafikus szimbólum is használható. A [SHIFT] és [C=] billentyű egyidejű lenyomása utána a nagybetű/kisbetű üzemmódba jutunk. Az elnevezés ebben az esetben nem egészen pontos, mivel még ekkor is rendelkezésünkre áll néhány grafikus szimbólum. Ezek a billentyűk elülső oldalának bal felén feltüntetett grafikus jelek, és a [C=] billentyű segítségével érhetjük el ezeket.

A nagybetű/kisbetű üzemmódban a begépelt szöveg alapértelmezésben kisbetűvel, [SHIFT]-tel együtt pedig nagybetűvel jelenik meg.

A [C=] billentyű (a számbillentyűkkel együtt) további nyolc rajzolószín előírására ad lehetőséget. A színbillentyű hozzárendelést az 5. fejezet tartalmazza.

A COMMODORE 64-es alaphelyzetbe állítása

Ha a tv-készülék beállításához használt színminta még a képernyőn van, akkor nyomjuk le a [SHIFT] és [CLR/HOME] billentyűt! Ekkor a képernyő törlődik, a kurzor pedig a bal felső sarokba kerül.

Ezután állítsuk vissza a normál világoskék rajzolószínt! Ezt a [C=] és a [7]-es billentyű egyidejű megnyomásával tehetjük meg. Mivel számítógépünk még mindig inverz video üzemmódban van – erre a színcsíkok előállításához volt szükségünk –, a [CTRL] és [0] billentyű lenyomásával térjünk vissza a normál kijelzéshez!

(A továbbiakban az [A] [B] rövidítést használjuk, ha az [A] és [B] bilentyűt egyidejűleg kell lenyomnunk.)

Az alaphelyzet visszaállításának van egy lényegesen egyszerűbb módja is:

[RUN/STOP] [RESTORE]

Ez a parancs néhány fontos regiszterben beállítja a kezdeti értéket, az éppen a tárban levő programot viszont érintetlenül hagyja. Ha gépünket a bekapcsolás után az automatikusan előálló kezdeti állapotba szeretnénk állítani, akkor gépeljük be a következő sort:

```
SYS 64738
```

majd nyomjuk le a [RETURN] billentyűt! Bánjunk óvatosan ezzel a paranccsal, mivel ilyenkor minden tárban levő program és adat törlődik.

Programok betöltése és tárolása

A COMMODORE 64-es egyik legfontosabb sajátsága, hogy programjainkat
ettán, ill. mágneslemezen tárolhatjuk és a felhasználás pillanatában újra a tárba
:hetjük. Ellenőrizzük, hogy a kazettásegység, ill. a hajlékonylemezes meghajtó
yesen van-e csatlakoztatva!

·ogramok töltése a tárba

1. **Modul:** A kereskedelem a modulokba integrált játék-, üzleti stb. programok széles skáláját kínálja.

   A MODULOK BEDUGÁSA ÉS KIVITELE ELŐTT MINDIG KAPCSOLJUK KI A SZÁMÍTÓGÉPET!!!

   Ha nem tartjuk be ezt a szabályt, akkor mind a modul, mind a számítógép tönkremenetelét érhetjük el.
   Miután bedugtuk a modult, kapcsoljuk be ismét a gépet, majd indítsuk el a programot a modulleírásban megadott paranccsal, ill. start billentyűvel!

2. **Kazetta:** Tegyük a programot tartalmazó kazettát a kazettásegységbe, és ha szükséges tekerjük vissza a szalagot a kazetta elejére! Ezután írjuk be:

   LOAD

   majd nyomjuk le a RETURN billentyűt! A LOAD (betöltés) parancsra a gép a következő üzenettel válaszol a képernyőn:

   PRESS PLAY ON TAPE (Nyomja le a lejátszásgomot!)

   Az utasítást követve nyomjuk le a kazettásegységen a PLAY-jel jelölt gombot! Ekkor a felirat és a kurzor eltűnik a képernyőről, a szalag pedig forogni kezd. A számítógép egy idő múlva újabb üzenettel jelentkezik:

   FOUND programnév (megtalálta programnév)

   Ha most lenyomjuk a C= billentyűt, akkor a megtalált program a COMMODORE 64-es tárába töltődik. A betöltési folyamatot a RUN/STOP billentyűvel bármikor megszakíthatjuk.

3. **Hajlékonylemez:** Nyissuk fel a meghajtóegység ablakát, és a lemezt dugjuk a nyílásba a címkés oldalával felfelé úgy, hogy a címke a lemez hozzánk közelebbi felén legyen! Minden lemezen található egy kis négyzet alakú bevágás (ese-

tenként ragasztószalaggal befedve). Ha mindent jól csináltunk, akkor ez a bevágás most a lemez bal oldalán van. Ha a lemez a helyén van, csukjuk le óvatosan a meghajtó ajtaját, és írjuk be:

```
LOAD"PROGRAMNÉV",8
```

majd nyomjunk RETURN-t!
Hallani fogjuk, hogy a meghajtóegység elindul, és a képernyőn a következő felirat tűnik fel:

SERCHING FOR PROGRAMNÉV (keresem...)
LOADING (betöltöm)
READY (kész)

Miután a READY felirat megjelent, a programot a RUN parancs beírása után RETURN leütésével indíthatjuk.

Programok betöltése szalagról

Ha a szalagról azt a programot szeretnénk a tárba tölteni, amelyet korábban PROGRAMNÉV alatt tároltunk, akkor tekerjük vissza a kazettát, majd írjuk be:

```
LOAD"PROGRAMNÉV"
```

Ha közben már elfelejtettük a program nevét, akkor csak a LOAD parancsot gépeljük be, majd nyomjuk le a RETURN-t! Ebben az esetben a szalagról automatikusan az első program töltődik a tárba. A számítógép a következő üzenetet írja a képernyőre:

```
PRESS PLAY ON TAPE
```

Ha követjük az előírást, akkor a felirat és a kurzor eltűnik, a gép pedig keresni kezdi a kiválasztott programot. Miután megtalálta újra bejelentkezik a képernyőn:

```
FOUND PROGRAMNÉV
```

A betöltés engedélyezéséhez ekkor meg kell nyomnunk a C= gombot. Miután a betöltés véget ért, újra megjelenik a kurzor és a READY felirat.

Programok betöltése hajlékonylemezről

Ha egy programot lemezről szeretnénk betölteni, hasonlóképpen kell eljárnunk, mint a szalag esetében. Írjuk be:

```
LOAD"PROGRAMNÉV",8
```

A RETURN lenyomása után a képernyőn a következőket láthatjuk:

```
SERCHING FOR PROGRAMNÉV
LOADING
READY
```

Megjegyzés: Amikor egy programot a gépbe töltünk, akkor minden korábban a :ban levő adat törlődik. Ezért az új program betöltése előtt ne feledkezzünk meg ·égi tárolásról!

rogramok tárolása szalagra

Ha egy általunk megírt programot szalagon szeretnénk tárolni, akkor a következ t kell beírnunk:

SAVE"PROGRAMNÉV" (tárold)

ol a PROGRAMNÉV egy általunk szabadon választott, maximálisan 16 karakter sszúságú elnevezés programunk azonosítására. A RETURN lenyomása után a

PRESS PLAY AND RECORD ON TAPE
(Nyomjuk le a lejátszás és felvétel gombot!)

enet jelenik meg.

Ha követjük az utasítást, akkor a felirat és a kurzor eltűnik a képernyőről és a p csak a tárolás befejezése után fogad el újabb utasítást.

rogramok tárolása hajlékonylemezre

A követendő eljárás talán még egyszerűbb, mint a szalag esetében. Írjuk be:

SAVE"PROGRAMNÉV",8

A 8 a meghajtóegység készülékszáma. Megadása arról tájékoztatja a gépet, gy a programot lemezen akarjuk tárolni. A RETURN lenyomása után a)MMODORE 64-es a

```
SAVING PROGRAMNÉV
OK
READY
```

enettel tájékoztat a tárolás sikeres teljesítéséről.

A PRINT parancs és számolás a COMMODORE 64-essel

Miután megismerkedtünk a betöltési és tárolási műveletekkel, hozzáfoghatunk első programunk megírásához.

PRINT"SOK SIKERT A COMMOODORE 64-ESHEZ" Gépeljük be ezt a sort és nyomjunk RETURN -t!

SOK SIKERT A COMMODORE 64-ESHEZ Ezt már a gép írja ki

Ha a beírás során hibát követtünk el, használjuk az INST/DEL billentyűt a kurzortól balra eső karakter törlésére! Így tetszőleges hosszúságú részt törölhetünk a beírt szövegből.

Nézzük meg még egyszer közelebbről, hogy mit is csináltunk pontosan! Először egy parancsot (PRINT) írtunk be, amely a gépet az idézőjelbe foglalt szöveg képernyőre írására utasította. A gép a RETURN lenyomása után felismerte és végrehajtotta a parancsot, azaz a képernyőre írta a

SOK SIKERT A COMMODORE 64-ESHEZ

szöveget.

Ha a gép a

? SYNTAX ERROR (szintaxis hiba)

üzenettel jelentkezik, akkor valamit rosszul csináltunk, pl. elvétettünk egy betűt a parancsban vagy elhagytunk egy idézőjelet. A gép "szőrszálhasogató", kizárólag azokat a parancsokat ismeri fel, amelyek megfelelnek az előírt sémáknak. Ennek ellenére ne féljünk a kísérletezéstől, hiszen a hibás parancsok nem tehetik tönkre a gépet. A tanulás legjobb módszere, ha mindent kipróbálunk, és megfigyeljük, hogy mi történik. A PRINT az egyik legfontosabb és leghasznosabb parancs. Segítségével bármit a képernyőre írhatunk, beleértve a grafikus szimbólumokat és a számítások eredményeit is. Próbáljuk ki a következő példát! (Előtte töröljük a képernyőt SHIFT CLR/HOME -mal!)

PRINT 12+12 Írjuk be ezt a sort és nyomjuk le a RETURN -t!

24 A gép válasza.

READY

Ebben a példában a COMODORE 64-est egyszerű számológépként használtuk. ; elvégzendő számítást egy PRINT parancs előzte meg. Minden olyan számítási ïveletet végrehajthatunk, amely számítógépünkön rendelkezésre áll: összeadás, ,onás, szorzás, osztás, hatványozás, gyökvonás, trigonometrikus műveletek stb.
Próbáljuk ki mi történik, ha a 12+12-t idézőjelek közé írjuk!

z összeadás

Az összeadást a (+) jel használatával végezzük. Ne felejtsük el, hogy a gép a ïveletet csak a RETURN billentyű lenyomása után tudja végrehajtani.

kivonás

A kivonáshoz a szokásos (–) jelet használjuk.

```
PRINT 12-9                RETURN
3
```

szorzás

A szorzás műveleti jele a csillag (∗).

```
PRINT 12*12               RETURN
144
```

z osztás

Az osztást a (/) jellel hajtjuk végre.

```
PRINT 144/12              RETURN
12
```

hatványozás

A hatványozás jele (↑) elé a hatványalapot, mögé pedig a kitevőt kell írnunk.

```
PRINT 12 ↑ 5
248832
```

A következő parancs ugyanezt az eredményt adja:

```
PRINT 12*12*12*12*12
248832
```

Tanács:

A BASIC parancsok helyett rövidítésüket is használhatjuk. Ezeket a D Függelék foglalja össze. A PRINT parancs rövidítése pl. a kérdőjel (?).

Próbáljuk ki a (?) rövidítést a következő példában!

? 3+5-7+2	A (?) a PRINT parancsot
3	helyettesíti.

Eddig csak egyszerű műveleteket végeztünk kis számokkal, a COMMODORE 64-es azonban ennél lényegesen többre képes. A következő példában nagy értékű tizedestörteket adunk össze. Jegyezzük meg jól, hogy a tizedestörtek felírásához a megszokott tizedesvessző (,) helyett a (.) pontot kell használnunk.

```
? 123.45 + 345.78 + 7895.687
8364.917
```

Az eredmény helyesnek látszik, végezzünk el azonban egy újabb számítást:

```
? 12123123.45 + 345.78 + 7895.687
12131364.9
```

Ha vesszük a fáradságot és papíron ceruzával is kiszámítjuk ezt az összeadást, eltérő eredményre jutunk. Vajon mi az oka ennek?

A COOMMODORE 64-es számítási pontossága nagy, de mégis korlátozott. A gép tíz számjeggyel számol, de az eredményt csak kilenccel jelzi ki.

A példában a gép ezért kerekítette az eredményt. A kerekítés a szokásos módon történik: ha a kerekítendő számjegy ötnél kisebb, akkor lefelé, ha öt vagy ötnél nagyobb, akkor felfelé.

A COMMODORE 64-es a 0.01 és 999999999 közé eső számokat a megszokott formában jelzi ki. Az e tartományon kívüli számok megjelenítésére a matematikai normálalakot használja. A matematikai normálalak a számokat mint a tíz hatványait ábrázolja. Írjuk be a következő példát:

```
? 123000000000000000
1.23E+17
```

Az E betű mögötti szám az amelyre a tízet kell emelni. Így

1.23E+17 = 1.23 * 10 ↑ 17

Még ebben az ábrázolási módban is megvannak azonban azok a határok, amelyek között a gép számolni képes. A legnagyobb abszolut értékű szám:

1.70141183E+38

A legkisebb abszolút értékű szám:
2.93873588E-39 (az ennél kisebb abszolút értékű számokat a gép nullának tekinti.)

Az elsőbbségi szabályok

Könnyen előfordulhat, hogy miközben összetettebb műveleteket próbálunk égezni, nem a várt eredményt kapjuk. Ezért feltétlenül meg kell ismernünk azokat szabályokat, amelyek alapján gépünk az összetett matematikai kifejezéseket rtékeli.
Vizsgáljuk meg a következő egyszerű műveletet:

20+8/2

Attól függően, hogy az összeadást vagy az osztást végezzük el elsőként, a eredmény 14 vagy 24 lehet. Ha a COMMODORE 64-essel számolunk, 24-et punk eredményül, mivel a gép először az osztást végzi el. A különböző aritmetikai iveleteket ún. osztályokba soroljuk, és a számítógép először a magasabb osztályba tozó műveletet hajtja végre.

1. osztály (legmagasabb):	(–) negatív előjel
2. osztály :	(↑) hatványozás
3. osztály :	(*,/) szorzás, osztás
4. osztály :	(+,–) összeadás, kivonás

Az automatikus kiértékelési sorrendet zárójelek elhelyezésével módosíthatjuk. A ójelezés hatását a következő példával szemléltetjük.

```
? 35/5+2
9
```

A gép itt a 35-öt először elosztja 5-tel (=7), majd az így kapott 7-hez hozzáadja -t, és így a végeredmény 9 lesz.
Ha az 5+2-t zárójelbe írjuk:

```
? 35/(5+2)
5
```

akkor a COMMODORE 64-es először a zárójelbe tett összeadást végzi el (=7), ıjd a 35-öt elosztja 7-tel (az összeadás eredményével).
A következő példában a gép először a zárójelben álló összeadásokat számítja ki, ıjd az így kapott számokat (21 és 7) összeszorozza:

```
? (12+9)*(6+1)
147
```

A PRINT parancs idézőjellel és idézőjel nélkül

Mindeddig számítógépünk a matematikai műveleteknek csupán az eredményét írta ki a képernyőre. Az idézőjelek megfelelő elhelyezése lehetővé teszi, hogy a kitűzött feladat és az eredmény együtt jelenjen meg. Gondoljunk arra, hogy bármi, ami idézőjelben áll, minden változás nélkül kerül kijelzésre, míg az idézőjelbe nem tett műveletek csupán az eredményt jelenítik meg! A következő példában gépünknek ezt a sajátságát használjuk ki. A záró idézőjel után tett pontosvessző hatása az, hogy a számítási eredmény közvetlenül a felírt művelet után jelenik meg a képernyőn.

```
? "5*9=";5*9
 5*9=45
```

A beírt sor első látásra kissé érthetetlennek tűnik. Az első rész: "5*9" a kitűzött feladatot írja ki, míg a pontosvessző (;) utáni rész a tényleges számítási előírás, amelynek eredménye szintén megjelenik a képernyőn.

Korántsem mindegy azonban, hogy milyen jellel választjuk szét a PRINT parancs részeit. Próbáljuk ki mi történik, ha a pontosvessző helyére vesszőt írunk!

Láthatjuk, hogy míg pontosvessző esetén az eredmény közvetlenül a művelet mögé íródott, addig a vessző hatására az eredmény távolabb került az egyenlőségjeltől.

A vessző hatásának pontos megértéséhez írjuk be a következő példát!

```
? 2,3,4,5,6                          RETURN
2          3          4          5
6
```

A COMMODORE 64-es az első négy jelet "egyenlően szétosztotta" az első sorban az ötödiket pedig áttolta a következő sorba. Ennek oka az, hogy számítógépünk a képernyőt négy 10 karakter hosszúságú szakaszra tagolja. A vessző egy ún. tabulátor, amelynek hatására a kijelzés mindig a következő szakasz elejére ugrik, még akkor is, ha az a következő sorban van.

Ha kellően tudatosítjuk magunkban a vessző és a pontosvessző hatása közötti különbséget, akkor a bonyolultabb kijelzési formák megtervezése nem fog különösebb nehézséget okozni.

3. FEJEZET

BASIC PROGRAMOZÁSI ALAPISMERETEK

GOTO
Szerkesztés
IF...THEN
FOR...NEXT

GOTO

Az eddigiekben csupán egyszerű műveleteket hajtottunk végre, miközben mindig csak egyetlen parancssort vittünk a gépbe. A RETURN billentyű lenyomása után a számítógép azonnal végrehajtotta az illető parancsot. Ezt a működési módot DIREKT üzemmódnak nevezzük.

A COMMODORE 64-es azonban ennél jóval többre képes. Nagyszámú parancssort tud közvetlenül egymás után feldolgozni, azaz képes egy PROGRAMOT végrehajtani.

Rögtön bemutatjuk, milyen egyszerű a programkészítés a COMMODORE 64-esen:

Töröljük a képernyőt SHIFT CLR/HOME-mal!

Ezután írjuk be a

```
NEW
```

parancsot, és nyomjuk le a RETURN-t! Ezzel törlünk a programtárból minden programot, amit eddigi kísérletezgetésünk során esetleg a gépbe vittünk.

Írjuk most be a következőket, és ne felejtsünk el mindkét sor után RETURN-t nyomni!

```
10 ?"COMMODORE 64"
20 GOTO 10      (ugrás)
```

Végezetül írjuk be a

```
RUN
```

parancsot, és "küldjük el" RETURN-nel! Mi történik most? A képernyő megelevenedik. Ha kigyönyörködtük magunkat, szakítsuk meg a program futását a RUN/STOP billentyűvel!

Már ebben a rövid példában is felhasználtunk néhány olyan dolgot, amelyek alapvető fontosságúak a programozásban. Figyeljük meg, hogy minden utasítást egy sorszám előz meg. Ezek a sorszámok arról tájékoztatják a gépet, hogy az egyes utasításokat milyen sorrendben hajtsa végre. Ha a program futása közben egy meghatározott pontra szeretnénk ugrani (a példában a GOTO 10 utasítás), akkor a sorszámokat jelzőpontként használhatjuk. A sorszámok a 0 és a 63999 közötti egész számok lehetnek.

```
10 PRINT"COMMODORE 64"
↑  ↑
|  └── utasítás
└───── sorszám
BREAK IN 10     (a 10. sorban megszakítás)
```

A sorszámozásnál célszerű tízesével haladni, mivel ez leegyszerűsíti az új

ɔgramsorok későbbi beszúrását. Példánkban a PRINT mellett egy újabb BASIC ṙancs is előfordult, a GOTO. A GOTO-val arra utasítjuk a gépet, hogy a GOTO ɩgé írt sorra ugorva folytassa a program végrehajtását.

```
10 ? "COMMODORE 64"
20 GOTO 10
```

Programunk tehát a következőképpen működik:

A 10. sorban a PRINT utasításnak megfelelően kiírja az idézőjelbe foglalt ɩveget, majd továbblép a 20. sorra. Mivel itt azt az utasítást találja, hogy ɔrjon a 10. sorra, a vezérlés újra a PRINT parancsra kerül. Ilyen módon a gép ẏamatosan nyomtatja a képernyőre a COMMODORE 64 szöveget, míg a program ását a [RUN/STOP] billentyűvel félbe nem szakítjuk.

A program beállítása után írjuk be a

LIST (lista)

ṙancsot és nyomjunk [RETURN]-t! Láthatjuk, hogy programunk változatlanul a ban van. Vegyük észre, hogy a gép a (?)-et PRINT-re cserélte. A program most ·dosítható, tárolható vagy újra indítható.

(A direkt üzemmódban a parancssor csak addig "él", amíg a képernyőn van; ha ĩnik, akkor a LIST paranccsal sem hozható vissza.)

A szerkesztés

Ha egy programsor beírásakor hibát követnénk el, akkor a következő javítási etőségeink vannak:

1. Újra beírhatjuk az adott sort (ugyanazzal a sorszámmal); a régi sor ilyenkor automatikusan az újjal helyettesítődik.
2. Ha beírjuk a hibás sor számát és lenyomjuk a [RETURN]-t, akkor az illető sor törlődik.
3. A kívánt sort a kurzorvezérlő és szerkesztő billentyűkkel tetszésünk szerint módosíthatjuk.

Tegyük fel, hogy egy programsor beírásakor hibát követtünk el. Vigyük a ·zort a [SHIFT] és a [CRSR] billentyűk segítségével a hibás sorra! A soron ĩl a [SHIFT] és a [CRSR] billentyűkkel mozoghatunk. Álljunk a kurzorral ɩibás karakterre, majd írjuk át a helyesre! Ha most [RETURN]-t nyomunk, ɩor a kijavított sor a programtárba kerül; erről a program kiíratásával (LIST) ẓződhetünk meg.

Ha egy adott helyre egy újabb karaktert szeretnénk beszúrni, akkor vigyük a

kurzort a kívánt helyre, majd nyomjunk SHIFT INST/DEL -t! Ezzel létrehozunk egy üres helyet, ahová a hiányzó karakter beírható. Ha az INST/DEL billentyűt SHIFT nélkül nyomjuk le, akkor a kurzor előtti karakter törlődik anélkül, hogy a helyén üres hely maradna.

Az egyes sorokat tetszőleges sorrendben módosíthatjuk; a gép a sorszám alapján dönt az illető sor elhelyezéséről a programban.

Változtassuk meg példaprogramunk 10-es sorát a következők szerint. (Vegyük észre a sor végén a (;) pontosvesszőt!)

```
10 PRINT"COMMODORE ";
20 GOTO 10
```

Indítsuk el RUN-nal a programot! Ha a RUN/STOP billentyűvel megszakítjuk a futtatást, a képernyőn várhatóan a 4. oszlopban megjelenik az idézőjelben levő szöveg.

Változók

Bármely programozási nyelv egyik legfontosabb fogalma a változó. Ezért most megismerkedünk a változók tulajdonságaival és a lehetőségekkel.

Képzeljük el, hogy számítógépünk egy halom dobozt tartalmaz, amelyekben számok vagy betűk vannak. Minden doboznak van egy általunk tetszés szerint választott neve. Ezeket a neveket változóknak nevezzük. A változók reprezentálják azokat az adatokat, amelyek egy adott pillanatban az illető dobozokban vannak. Nézzük a következő példát:

```
10 X%=15
20 X=23.5
30 X$="X% ES X ÖSSZEGE="
```

A gép a következőképpen tölti fel az egyes változókat:

```
X% 15
X 23.5
X$ X% ES X ÖSSZEGE=
```

Egy változó egy adott tárhelyet azonosít. A változó értéke (azaz a tárhely tartalma) a program futása közben változhat. Ahogy láttuk, a változókhoz egész számokat, tizedestörteket vagy szöveget is rendelhetünk.

Ha egy változó neve mögé a (%) jelet írjuk, akkor az adott változó csak egész számokat hordozhat. Az ilyen változókat egész típusúnak (integertípus) nevezzük. A következők mind egész típusú változók:

```
A%
X%
A1%
NM%
```

Ha a változó nevét a ($) jel követi, akkor karakterfüzér (string) típusú változóról ı szó, amelyhez tetszőleges karaktersorozatot rendelhetünk. Ez a karakterfüzér ;zőleges betűket, számokat, ill. különleges jeleket is tartalmazhat. Az érvényes akterfüzér változónevek:

```
A$
X$
MI$
```

A tizedestörteket (lebegőpontos számokat) pl. a következő változónevek képvi-ıetik:

```
A1
X
Y
MI
```

Noha egy változó neve több karakterből is állhat, a számítógép a változókat a 'nek csupán az első két karaktere alapján azonosítja. Ezért nem tud különbséget ni pl. a KOLLÉGA és KONCERT nevű változók között. A név első karaktere k alfanumerikus karakter (A és Z közötti betű), a második pedig betű vagy szám et. Az utolsó jel mindig a változó típusát jelzi:

% = egész szám
$ = karakterfüzér

Ha a név végén sem (%) sem ($) jel nem áll, akkor a változó lebegőpontos ısú.
A változónevek nem tartalmazhatnak BASIC kulcsszavakat (azaz olyan szavakat, elyeknek a BASIC nyelvben rögzített jelentése van). Ezért a pl. a ROTOR név n elfogadható mivel magában foglalja a TO BASIC kulcsszót. (A foglalt kulcs-vakat a D Függelék foglalja össze.)
Gépeljük be most a következő programot, majd indítsuk el a RUN paranccsal! ; felejtsünk el minden sor után RETURN -t nyomni!)

```
NEW

10 X%=15
20 X=23.5
30 X$="X% ES X OSSZEGE : "
40 PRINT "X%="; X%, "X="; X
50 PRINT X$; X%+X

READY.
```

Ha mindent jól csináltunk, akkor a program lefuttatása után a következő áll a képernyőn:

```
RUN
X% = 15 X = 23.5
X% ES X OSSZEGE = 38.5
READY
```

A 10-es és 20-as sorban az X% és X változókhoz sorban a 15 és 23.5 értéket rendeltük. Az X$ karakterfüzért a 30-as sorban definiáltuk. A 40-es sorban a PRINT utasítás különböző formáit kombináltuk az X% és X változók nevének és értékének kiíratásához. Az 50-es sor az X$ karakterfüzért és az X%+X összeget írja ki.

Noha mindhárom változónév X-szel kezdődik, a gép a (%), illetve a ($) jel alapján meg tudja különböztetni egymástól a három változót.

Ha egy változó értékét megváltoztatjuk, akkor a változó "dobozba" az új érték kerül. Vizsgáljuk meg ennek alapján a következő értékadó utasítást:

```
X = X + 1
```

Ez az egyenlőség a matematikában értelmetlen, a programozásban mégis gyakran használjuk. Hatása a következő: vedd az X változó értékét, adj hozzá egyet, majd az eredményt tedd vissza X-be!

IF ... THEN (ha ... akkor)

A változókról szerzett ismereteinket rögtön kamatoztathatjuk is a következő programban:

```
10 SZ=0
20 PRINT "COMMODORE 64"
30 SZ=SZ+1
40 IF SZ<5 THEN 20
50 END

READY.
```

```
RUN
COMMODORE 64
COMMODORE 64
COMMODORE 64
COMMODORE 64
COMMODORE 64
```

Láthatjuk, hogy az IF ... THEN utasítás használatával meghatározott számú sort írattunk a képernyőre.

a a gép a program futása közben egy IF utasításba "botlik", akkor megvizs- hogy az IF mögött álló feltétel igaz-e. Ha igaz, akkor végrehajtja a THEN tt álló utasítást. Ha nem, akkor a végrehajtás a következő sorral folytatódik. ' ... THEN utasításban a következő feltételek teljesülését vizsgálhatjuk:

SZIMBÓLUM	JELENTÉS
<	kisebb, mint
>	nagyobb, mint
=	egyenlő
<>	nem egyenlő
>=	nagyobb vagy egyenlő
<=	kisebb vagy egyenlő

:zekkel a feltételekkel egyszerűen, mégis hatékonyan vezérelhetjük program- : futását.

```
  10 SZ=0
- 20 ?"COMMODORE 64"
  30 SZ=SZ+1
- 40 IF SZ < 5 THEN 20
  ↓
  50 END
```

Példaprogramunkban egy ún. ciklust építettünk fel, amelyben a program addig ózkodik, amíg az IF és a THEN közötti feltétel teljesül. A 10-es sorban az SZ ısszámlálót 0-ra állítjuk. A 20-as sor írja ki a COMMODORE 64 szöveget. A s sorban a ciklusszámláló értékét a ciklus minden egyes lefutása után 1-gyel :ljük. A ciklusszámláló aktuális értéke tehát azt jelzi, hogy hányszor futott már ciklus. A 40-es sorban vizsgáljuk meg a vezérlési feltételt. Ha SZ értéke kisebb l (azaz a szöveget 5-nél kevesebbszer nyomtattuk ki), akkor a program visszatér -as sorra és kiírja a COMMODORE 64 szöveget. A ciklus 5. végigfutása után ltétel többé nem teljesül, és a program az 50-es sorra lép. Itt az END (vége) ítás áll, amely ebben a példában esetleg el is hagyható. SZ felső határának osításával tetszőleges számú szövegsort írhatunk a képernyőre.

FOR ... NEXT CIKLUS (... következő)

Az előző példában megvalósított ciklust a FOR ... NEXT utasításokkal egyszerűbb módon is létrehozhatjuk. Gépeljük be, majd indítsuk el a következő programot:

```
10 FOR SZ=1 TO 5
20 PRINT "COMMODORE 64"
30 NEXT SZ

READY.
```

```
RUN
COMMODORE 64
COMMODORE 64
COMMODORE 64
COMMODORE 64
COMMODORE 64
```

Ugyanazt a képet kapjuk, mint az előző esetben, noha programunk most jóval rövidebb.

A 10-es sorban SZ-t (ciklusszámláló) 1-re állítjuk. A 20-as sor újra a kiírást végzi, a 30-as pedig a ciklusszámláló értékét növeli 1-gyel. A 30-as sorban álló NEXT utasítás hatására a program az ehhez a NEXT-hez tartozó (10-es) sorra ugrik. A FOR ... NEXT ciklus annyiszor kerül végrehajtásra, amennyi a 10-es sorban a TO (ig) kulcsszó után álló érték.

Ha a 10-es sorba semmi mást nem írunk, akkor a ciklusszámláló értéke minden lefutás után 1-gyel nő. Más "lépésközt" is előírhatunk azonban:

```
10 FOR SZ=1 TO 10 STEP .5
20 PRINT SZ,
30 NEXT SZ

READY.
```

```
RUN
 1          1.5         2          2.5
 3          3.5         4          4.5
 5          5.5         6          6.5
 7          7.5         8          8.5
 9          9.5         10
```

Ebben a példában a STEP (lépés) kulcsszó után álló szám értéke határozza ; a lépésközt. Ebben az esetben az SZ (szám) értéke minden lefutás után 0.5-nő (a tizedestörtek esetén a 0 elhagyható). Ilyen módon ezzel a programmal számsorozatot írhatunk a képernyőre, amelyben a számok értéke rendre 0.5-del ekszik.

Negatív lépésközt is választhatunk. Próbáljuk ki, hogy mi történik, ha a 10-es a következőképpen módosítjuk:

```
10 FOR SZ = 10 TO 1 STEP -.5
```

4. FEJEZET

TOVÁBBI BASIC UTASÍTÁSOK

Bevezetés
Egymásba ágyazott ciklusok
INPUT
GET
Véletlenszám-generálás
Kitalálós játék
Kockajáték
Véletlengrafika

Bevezetés

A következő fejezeteket elsősorban azoknak ajánljuk, akik már szereztek némi tapasztalatot a BASIC nyelv használatában. Olyan módszereket mutatunk, amelyek ismerete a haladó programozók számára feltétlenül szükségesek.

A kezdők e fejezetek egyes részeit esetleg túlságosan szakmainak találják. Nekik sem szabad azonban elkedvetlenedni, mivel a SPRITE és a HANGGENERÁLÁS című fejezetben számukra is elhelyeztünk néhány szórakoztató példát. Ezekben betekintést adunk a COMMODORE 64-es kiváló grafikai és hangképző tulajdonságaiba.

Mozgás a képernyőn

A következőkben az eddig tanultakat fogjuk néhány új módszerrel kiegészíteni. Írjuk be a következő programot, amelynek az az érdekessége, hogy a kurzor vezérlésére a PRINT utasítást használja! Ha a programban a kurzort pl. balra szeretnénk mozgatni, akkor beíráskor a SHIFT és a CRSR billentyűket kell lenyomnunk. Ennek hatására a képernyőn egy grafikus karakter jelenik meg. Az R Függelék megkönnyíti a kurzormozgató utasítások grafikus megfelelői közötti eligazodást.

Ha helyesen írtuk be a programot, akkor elindítás után egy, a képernyő bal és jobb széle között ide-oda pattogó labdát látunk.

```
10 REM PATTOGO LABDA
20 PRINT "■"
25 FOR X=1 TO 5: PRINT "■" : NEXT X
30 FOR LB=1 TO 40
40 PRINT " ■■"; : REM LABDA-SHIFT/Q
50 FOR ID=1 TO 5
60 NEXT ID
70 NEXT LB
80 FOR LB=40 TO 1 STEP -1
90 PRINT " ■■■■■";
100 FOR ID=1 TO 5
110 NEXT ID
120 NEXT LB
130 GOTO 20

READY.
```

A (:) kettőspont egy új utasítás kezdetét jelzi.

A betűközök feltétlenül szükségesek!

A működés bemutatásához a következő ábrán a program szerkezetét magyarázó nyilakat helyeztünk el.

A 10-es sorban a REM (REMark=megjegyzés) utasítás után a program neve áll.

```
10 REM PATTOGO LABDA
20 PRINT "▚"
25 FOR X=1 TO 5: PRINT "▚" : NEXT X
30 FOR LB=1 TO 40
40 PRINT " ●▌"; : REM LABDA-SHIFT/Q
50 FOR ID=1 TO 5
60 NEXT ID
70 NEXT LB
80 FOR LB=40 TO 1 STEP -1
90 PRINT " ▌▌●▌";
100 FOR ID=1 TO 5
110 NEXT ID
120 NEXT LB
130 GOTO 20

READY.
```

REM utasítás magyarázatok elhelyezését teszi lehetővé, a program működésére ımiféle hatása sincs.

A 20-as sorban töröljük a képernyőt. A 25-ös sorba írt ciklus 10 pozícióval lejb viszi a kurzort, így helyezzük a labdát a képernyő közepére. Ha ezt a sort agyjuk, akkor a labda a képernyő felső szélén pattog. Figyeljük meg, hogy egy lus FOR és NEXT utasítása egyazon sorban is állhat! Ha egy sorba több BASIC ısítást is írunk, akkor (:) kettősponttal kell szétválasztanunk ezeket.

A 30-as sorban egy újabb ciklus fejrésze áll.

Ez a ciklus mozgatja a labdát balról jobbra, keresztül az egész 40 oszlop lességű képernyőn.

A 40-es sor több feladatot is ellát. Először egy betűköz karaktert helyez el, ely törli a labdát az előző helyzetből. Ezután felrajzolja a labdát az aktuális yre, majd a kurzort egy karakterrel balra viszi. Ez megteremti az előfeltételt a da mostani helyzetéből való majdani letörléséhez.

Az 50-es és 60-as sor "üres" ciklusa a labda mozgásának lelassítását végzi. élkül a labda olyan gyorsan pattogna, hogy alig-alig látnánk. A 70-es sorban 0-as sorban kezdődő ciklus befejező NEXT utasítása található. E ciklus minden es lefutásakor a labda a képernyőn egy kurzorpozícióval jobbra lép. A programrkezetből világosan kiderül, hogy itt két ciklust ágyaztunk egymásba.

Ha az egymásba ágyazott ciklusok módszerét használjuk, akkor nagy figyelmet fordítanunk arra, hogy az elsőnek megnyitott ciklust zárjuk le utoljára. Ha iklusokat tévedésből keresztezzük (ebben az esetben pl. felcseréljük a 60-as és 0-es sort), akkor a gép hibajelzést ad, mivel programunk értelmetlenné válik! báljuk ki! A 80-as és 120-as sorok közötti programrész a fordított mozgási amatot valósítja meg.

A 90-es sor azonban kissé eltér a 40-estől, mivel most a törlés után kétszer balra lépnünk az aktuális pozíció felrajzolásához. (Ennek oka az, hogy egy karakter ajzolása vagy törlése után a kurzor automatikusan egy pozícióval jobbra lép, és en az esetben ez ellentétes a labda kívánt mozgási irányával.)

Végezetül a program visszaugrik a 20-as sorra és minden kezdődik elölről.

Hogy még jobban megértsük a program működését, változtassuk meg egy kicsit a 40-es sort:

```
40 PRINT "o";
```

Indítsuk el a programot, és nézzük meg, hogy mi történik! Mivel a kurzorvezérlést elhagytuk, a labda "megsokszorozódik" és csak a visszapattanó labda törli ki a felesleges labdajeleket.

INPUT (bevitel)

Eddigi példáinkban a programok futását csak az egyes sorok átírásával tudtuk megváltoztatni. Az INPUT utasítás használatával futás közben adhatunk át adatokat a programnak.

A következő példában az INPUT utasítás működését szemléltetjük.

```
10 INPUT A$
20 PRINT "AZT IRTA BE, HOGY : "; A$
30 PRINT
40 GOTO 10

READY.
```

```
RUN
? COMMODORE 64                     Mi írtuk be
AZT IRTA BE HOGY: COMMODORE 64     A gép válasza
```

Amikor a program az INPUT utasításhoz ér, megáll és egy kérdőjelet ír a képernyőre. Ezzel jelzi, hogy adatbevitelre vár a billentyűzetről. Írjunk be egy tetszőleges karaktersorozatot, majd nyomjunk RETURNT -t! A gép ekkor ezt írja ki:

AZT IRTA BE, HOGY: az általunk a ? mögé beírt szöveg

Célszerű, ha az INPUT utasítással együtt egy üzenetet is a képernyőre írunk amely arról tájékoztat, hogy a gép milyen adatra vár. Az üzenet legfeljebb 3 karakter hosszúságú lehet. Az INPUT utasítás szintaxisa:

INPUT "ÜZENET"; Változó

ahol a változó mind numerikus, mind karakterfüzér típusú lehet. Ha nincs szükségün az üzenetre, akkor a következőt kell írnunk:

INPUT Változó

Megjegyzés: Előbbi példaprogramunkból a következőképpen léphetünk ki:

| RUN/STOP | | RESTORE |

A következő példaprogram átszámítást végez a Celsius és a Fahrenheit fokban megadott hőmérsékleti adatok között. Megírásakor megpróbáltunk minél többet alkalmazni az eddig tanultakból.

```
5 PRINT "♥"
10 PRINT "ATVALTAS FAHRENHEITBE VAGY CELSIUSBA ?";
12 PRINT "(F/C)": INPUT A$
20 IF A$="" THEN 10
30 IF A$="C" THEN 100
40 IF A$="F" THEN 50
45 GOTO 10
50 INPUT "BEVITEL CELSIUSBAN: ";C
60 F=(C*9)/5+32
70 PRINT C; "CELSIUS FOK= "; F; "FAHRENHEIT FOK"
80 PRINT
90 GOTO 10
100 INPUT "BEVITEL FAHRENHEITBEN: "; F
110 C=(F-32)*5/9
120 PRINT F; "FAHRENHEIT FOK= "; C; "CELSIUS FOK"
130 PRINT
140 GOTO 10

READY.
```

A 10-es sor INPUT utasításával együtt egy kérdést írunk a képepernyőre.

A 20-as, 30-as és 40-es sorban a kérdésre adott választ vizsgáljuk. Ha semmit sem írtunk be válaszként, azaz csak a | RETURN | billentyűt nyomjuk le, akkor a program visszatér a 10-es sorra, és újra kiírja a kérdést. Ha a válasz a C betű, akkor a 30-as sor egy ugrást hajt végre a 100-as sorra. Itt kezdődik ugyanis az a programrész, amely a Fahrenheitben megadott hőmérsékletet Celsiusra számítja át. F válasz esetén az 50-es sorra ugrunk. Minden más esetben visszatérünk a 10-es sorra és újra kiírjuk a bevezető kérdést. (Erről a 45-ös sor gondoskodik.) Ha helyesen írtuk be a programot, akkor elindítása után megkérdezi, hogy milyen átváltást szeretnénk végrehajtani. Ezután kéri az átszámítandó értéket, majd kiírja az eredményt.

Az átszámítási formulát bármely alapfokú fizikakönyvben megtalálhatjuk; átírása BASIC nyelvre magától értetődő. Az eredmény kiíratása után a program a 10-es sorra ugorva elölről kezdi a végrehajtást. A képernyő a megadott adatok bevitele után így néz ki:

```
RUN
ATVALTAS FAHRENHEITBE VAGY CELSIUSBA (F/C)
?C
BEVITEL FAHRENHEITBE: 32
32 FOK FAHRENHEIT = 0 FOK CELSIUS
ATVALTAS FAHRENHEITBE VAGY CELSIUSBA (F/C)
?
```

Ha elégedettek vagyunk a programmal, akkor a későbbi felhasználásra tároljuk a programot szalagra vagy lemezre.

GET (elvesz)

A GET utasítással anélkül olvashatunk be egy karaktert a billentyűzetről a tárba, hogy utána megnyomnánk a RETURN-t.

Az adatbevitel így sok esetben lényegesen meggyorsítható. A beolvasott karaktert a GET utasításban álló változóhoz rendeljük. A következő rutin (általánosan alkalmazható programrészlet) a GET használatát szemlélteti:

```
1 PRINT "♥"
10 GET A$ : IF A$="" THEN 10
20 PRINT A$;
30 GOTO 10

READY.
```

Nincs betűköz!

A program elindítás után rögtön törli a képernyőt. Ha lenyomunk egy billentyűt, akkor a 20-as sor hatására megjelenik a bevitt karakter. Ezután a GET utasítás újabb bevitelre vár. Figyeljük meg, a beolvasott jel csak a PRINT hatására jelenik meg a képernyőn.

A 10-es sor második utasítása nagyon fontos. Ezzel vizsgáljuk meg, hogy lenyomtunk-e egyáltalán valamilyen billentyűt. Erre azért van szükség, mert a GET (ellentétben az INPUT-tal, ahol a bivitelt RETURN-nel érvényesítjük) megszakítás nélkül továbblép, ha nem talál bevitt karaktert. A 10-es sorban ezért egy ciklust hoztunk létre, amelynek magjában azt is megvizsgáljuk, hogy lenyomtunk-e valamilyen billentyűt (IF A$=""). Próbáljuk ki, hogy mi történik, ha a 10-es sor második felét elhagyjuk:

```
10 GET A$
```

A programot RUN/STOP RESTORE -ral állíthatjuk le. A GET utasítást a mérséklet-átszámító programban is elhelyezhetjük. Ehhez a 10-es, a 20-as és a -es sort kell átírnunk a következőképpen:

```
10 PRINT "ATVALTAS FAHRENHEITBE VAGY CELSIUSBA";
12 PRINT "(F/C) ?"
20 GET A$ : IF A$="" THEN 20
40 IF A$<>"F" THEN 20
45 :

READY.
```

Véletlenszám-előállítás

A COMMODORE 64-es számos beépített függvénnyel rendelkezik, amelyeket ,ük leírásával hívhatunk.

Mind az oktató-, mind pedig a játékprogramok írása során előnyös lehet a etlenszámok alkalmazása. Próbáljuk meg pl. egy kockadobálás szimulálását! Erre élra a beépített RND-függvény (RaNDom=véletlen) kifejezetten alkalmas. Írjuk a következő programot:

```
10 FOR X=1 TO 10
20 PRINT RND(1),
30 NEXT

READY.
```

A futtatás után a képernyő a következőket mutatja:

```
.776433747          .417980108
.829277551          .809331095
.967128073          .426060658
.364234364          .770340712
.922876569          .658604244
```

Netán más számokat látunk? Nem csoda, hiszen ezek véletlenszámok. Próbálkoz-nk még néhányszor és győződjünk meg róla, hogy a számok valóban minden futás án változnak. Egyetlen közös sajátságuk, hogy mindegyik a 0–1 intervallumba eső edestört. (Megjegyzés: 0-t vagy 1-et soha nem kapunk eredményül és a tizedes-jek száma 9, valamint exponenciális kijelzés is lehetséges.)

Gondoljuk végig, hogy mit kell tennünk, ha a kockajátékoz 0 és 6 közé eső egész számokat akarunk előállítani! Első nekifutásra próbáljuk meg a véletlenszámoknak csupán a tartományát megváltoztatni! Ehhez a 20-as programsort így kell átírnunk:

```
10 FOR X=1 TO 10
20 PRINT 6*RND(1)
30 NEXT

READY.

RUN
 4.90711388              1.1252062
 3.29855115              2.30889658
  .313614802             1.43789027
 4.14914835              3.3676486
 4.7888016               4.87432929
```

A következő lépésben szabaduljunk meg a felesleges tizedesjegyektől! A COMMODORE 64-es erre a célra is tartogat egy lehetőséget, mégpedig az INT (INTeger=egész) függvényt:

```
10 FOR X=1 TO 10
20 PRINT INT(6*RND(1))
30 NEXT

READY.

RUN
1    5    5    3
0    1    4    2
3    1
```

Az INT függvény levágja a tizedestörtek tizedesjegyeit.

Most már csak egy lépés van hátra: azt kell elérnünk, hogy ne 0 és 5, hanem 1 és 6 közötti egészeket kapjuk. Módosítsuk ezért a 20-as sort a következőképpen:

```
20 PRINT INT(6*RND(1))>+1
```

Most már a megfelelő számokat kapjuk.

E módszerrel tetszőleges tartományba eső egész véletlenszámokat is elő tudunk állítani. A tartomány nagyságát az RND függvény szorzója adja. Az így kapott intervallumot bárhová eltolhatjuk az egész számok skáláján egy megfelelő választott egész szám hozzáadásával. A felhasználandó matematikai alak:

Szám = INT(Tartomány*RND(1))+ELTOLAS

Kitalálós játék

Használjuk fel a véletlenszámokról tanultakat egy összetettebb programban is: Ebben a példában néhány újabb programozási trükköt is alkalmaztunk. A ékprogram úgy indul, hogy megkérdezi a kitalálandó számok tartományát. után kezdhetjük a találgatást. A program minden próbálkozásunkat kommentálja. Kiírja hogy az általunk választott szám kisebb vagy nagyobb, mint a alálandó. Ezt programozástechnikailag úgy valósítjuk meg, hogy a számítógép al választott számot egy IF...THEN utasítással összehasonlítjuk a billentyűzetről olvasottal.

```
1 REM SZAMKITALALOS JATEK
2 PRINT "□"
5 INPUT "A SZAMOK FELSO HATARA "; HT
10 SZ=INT(HT*RND(1))+1
15 CG=0
20 PRINT "MEGVAN A SZAMOM"
30 INPUT "AZ ON TIPPJE"; TP
35 TG=TG+1
40 IF TP>SZ THEN PRINT "AZ EN SZAMOM KISEBB";
42 PRINT : GOTO 30
50 IF TP<SZ THEN PRINT "AZ EN SZAMOM NAGYOBB";
52 PRINT : GOTO 30
60 IF TP=SZ THEN PRINT "NAGYSZERU, KITALALTA A SZAMOT"
65 PRINT "MINDOSSZE"; TG; "PROBALKOZASSAL" : PRINT
70 PRINT "AKAR MEG EGYET JATSZANI ? (I/N)";
80 GET VA$ : IF VA$="" THEN 80
90 IF VA$="I" THEN 2
100 IF VA$<>"N" THEN 80

READY.
```

Próbáljuk meg úgy módosítani a programot, hogy a játékos a számoknak ne k a felső, hanem az alsó határát is megválaszthassa.
A TG változó értéke minden téves találgatás után 1-gyel növekszik. A program elyes tipp után:

NAGYSZERU, KITALALTA A SZAMOT szöveg mögé írja a megfejtést.

Tipp:

A 40-es és 50-es sorokba több parancsot is írtunk, kettőspontokkal szétválasztva. en módon nemcsak áttekinthetőbbé tehetjük a programot, hanem a programtárral akarékoskodhatunk.
Ugyanezekbe a sorokba kiegészítő PRINT utasításokat is elhelyeztünk, az üres ok szövegbe iktatásával áttekinthetőbbé tesszük a képernyőn megjelenteket.
Mivel kezdetben tízesével számoztuk a sorokat, később egyszerű volt a hibás álgatásokat számláló rutin beiktatása. (15-ös, 35-ös és 65 sor)

Kockajáték

A következő program egy kockajátékot szimulál, két kockával. Sok szerencsét! Gondoljuk végig az eddig tanultakat és próbáljuk megfejteni a program működését!

```
5 PRINT "PROBALJON SZERENCSET !"
10 PRINT "PIROS KOCKA= ";
15 PRINT INT(6*RND(1))+1
20 PRINT "FEHER KOCKA= ";
25 PRINT INT(6*RND(1))+1
30 PRINT "A KOVETKEZO DOBASHOZ NYOMJA";
32 PRINT "LE A BETUKOZ BILLENTYUT !" : PRINT
40 GET A$ : IF A$="" THEN 40
50 IF A$=CHR$(32) THEN 10

READY.
```

Véletlengrafika

A következő rövid példában azt mutatjuk be, hogy a véletlenszámokra grafikát is építhetünk:

```
10 PRINT "▛"
20 PRINT CHR$(205.5+RND(1));
30 GOTO 20

READY.
```

A lényeg a CHR$ (CHaracter String=karakterfűzér) függvényt tartalmazó 20-as sorban áll. A függvény nevét követő zárójelbe foglalt szám – 0 és 255 közé kell esnie – dönti el, hogy milyen grafikus szimbólum jelenik meg a képernyőn. A számok és karakterek összetartozását leíró táblázat az F Függelékben található.

A karakterek kódját azonban mi magunk is kideríthetjük az ASC (ASCII) függvény használatával. Írjuk be:

```
PRINT ASC("X")
```

X az a karakter, amelynek a kódjára kíváncsiak vagyunk. X tetszőleges karakter lehet, beleértve a grafikus szimbólumokat is. A kód, amelyről beszélünk ASCII kód néven ismert! Az ASC függvény a CHR$ fordítottja. Így ha az ASC-vel megállapított értéket (Y) a CHR$ függvénybe tesszük, akkor

```
PRINT CHR$(Y)
```

ɔeírása után az X-karaktert kapjuk vissza a képernyőn. Ha most beírjuk:

```
PRINT CHR$(205); CHR$(206)
```

.kkor az M és N billentyűk jobb oldalán levő grafikus szimbólumokat írjuk ki a :épernyőre. Ezekből a jelekből épül fel az az "útvesztő", amelyet példaprogramunk lindítása után látunk a képernyőn. A 205.5+RND(1) formulával 205.5 és 206.5 :özötti véletlenszámokat állítunk elő. Annak a valószínűsége, hogy egy generált zám kisebb vagy nagyobb, mint 205, ugyanaz. Mivel a CHR$ függvény nem 'eszi figyelembe a tizedesjegyeket, a 205-ös és 206-os kódokat elvileg ugyanazzal ı gyakorisággal állítjuk elő.

Ezt a valószínűséget és vele együtt az előálló képet úgy módosíthatjuk, hogy a :05.5-es kiindulási értéket néhány tizeddel csökkentjük vagy növeljük.

5. FEJEZET

GRAFIKA HALADÓKNAK

Színes grafika
Színbeállítás PRINT utasítással
CHR$ színkódok
PEEK és POKE
Újabb labdajáték

Szín és grafika

Megismerkedhettünk már a COMMODORE 64-es több fejlett szolgáltatásával. Az egyik legérdekesebb lehetőséget, a színes grafikát azonban még nem alkalmaztuk.

Mivel az eddigi példákban elsősorban gépünk számítási lehetőségeire helyeztük a hangsúlyt, mindeddig megelégedtünk az alap képernyőszínek használatával. Ebben a fejezetben a színkezelés és a grafikus sajátságok bemutatására helyezzük a hangsúlyt.

Színbeállítás PRINT utasítással

Bizonyára emlékszünk még az 1. fejezet útmutatásai alapján végrehajtott színtesztre. Akkor a képernyő színeit a [CRTL] és egy számbillentyű együttes lenyomásával módosítottuk.

Szükségünk lehet arra is, hogy a színeket a programból állítsuk be. Amint azt a "Pattogó labda" programban már láttuk, a billentyűzetről kiadható parancsokat is elhelyezhetjük programunkban.

A színeket egy 16 színből álló skálából választhatjuk. A [CRTL] és valamely számbillentyű együttes leütésével elérhető színek:

1: fekete 2: fehér 3: piros 4: türkiz 5: lila 6: zöld 7: kék 8: sárga

A C= és egy megfelelő számbillentyű együttes lenyomásával további 8 színt kaphatunk:

1: narancs 2: barna 3: halványpiros 4: sötétszürke 5: középszürke 6: világoszöld
7: világoskék 8: világosszürke

Írjuk be a NEW parancsot, majd csináljuk végig a következő példát!

A 10 PRINT" sorkezdet után nyomjuk le együtt a [CTRL] és az [1]-es gombot! Ezután írjuk be az [S] betűt! Most nyomjuk le együtt a [CTRL] és a [2]-es billentyűt, majd felengedésük után írjuk be a [P] betűt! Haladjunk végig ilyen módon a színskálán és a SPEKTRUM szó betűin!

```
10 PRINT" S   P   E   K   T   R   U   M"
          ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑
  [CTRL] [1] [2] [3] [4] [5] [6] [7] [8]
RUN
SPEKTRUM
```

A színvezérlő jelek a kurzorvezérlő jelekhez hasonlóan grafikus szimbólumok. A [CTRL] és a [3]-as billentyűk együttes lenyomásakor pl. a (£) font jel jelenik meg a képernyőn. A következő táblázat összefoglalja a színkódokat:

Billentyű	Szín	Kijelzés	Billentyű	Szín	Kijelzés
CTRL 1	fekete	▖	C= 1	narancs	♠
CTRL 2	fehér	E	C= 2	barna	╭
CTRL 3	piros	£	C= 3	halványpiros	╳
CTRL 4	türkiz	◥	C= 4	sötétszürke	○
CTRL 5	lila	▒	C= 5	középszürke	♣
CTRL 6	zöld	↑	C= 6	világoszöld	│
CTRL 7	kék	←	C= 7	világoskék	◆
CTRL 8	sárga	π	C= 8	világosszürke	┼

Ha lefuttatuk a programot, akkor észrevehettük, hogy a színvezérlő jelek csak ι listán láthatók, a SPEKTRUM szó már nélkülük jelenik meg a képernyőn (a negfelelő színvariációban) a futtatás után. Próbáljunk ki még néhány lehetőséget, ιogy egészen biztosak lehessünk a színkezelésben! (Ne feledkezzünk meg a C=)illentyűről sem!)

Megjegyzés: Ha egy program színvezérlést is tartalmaz, akkor a futtatás után ιz utolsóként beállított szín marad érvényben. A normál kijelzésre (világoskék, ötétkék) RUN/STOP RESTORE -ral térhetünk vissza.

CHR$ színkódok

Az F Függelékben (CHR$ kódok listája) azt láthatjuk, hogy a színek, akárcsak ι kurzorvezérlő parancsok, számkódokkal rendelkeznek. Ha ezeket a kódokat a ²RINT utasításban használjuk, akkor ugyanazt a hatást érjük el, mint a CTRL és ι megfelelő szám billentyű együttes lenyomásával. Próbáljuk ki a következő példát:

```
10 PRINT CHR$(147) : REM CLR/HOME
20 PRINT CHR$(30); "CHR$(30) ATSZINEZ?"

READY.

RUN
CHR$(30) ATSZINEZ
```

A szövegnek most zöldnek kell lennie. Sok esetben a színvezérlés a CHR$-iggvénnyel jóval egyszerűbb, mint színbillentyűkkel. A következő program színes síkokat rajzol a képernyőre. Figyeljük meg, hogy a 40–190-es sorok csak a orszámban és a CHR$-függvény argumentumában különböznek egymástól! Ezért zek közül elég csak egyet teljes terjedelmében beírni, a többit pedig a sorszám és . CHR$-függvény változtatásával állíthatjuk elő (lásd szerkesztési tanácsok).

```
1 REM SZINES CSIKOK
5 PRINT CHR$(147) : REM CHR$(147)=CLR/HOME
10 PRINT CHR$(18); "           "; : REM INVERZ CSIK
20 SZ=INT(16*RND(1))+1
30 ON SZ GOTO 40,50,60,70,80,90,100,110,120,130,140,";
32 PRINT 150,160,170,180,190
40 PRINT CHR$(5); : GOTO 10
50 PRINT CHR$(28); : GOTO 10
60 PRINT CHR$(30); : GOTO 10
70 PRINT CHR$(31); : GOTO 10
80 PRINT CHR$(144); : GOTO 10
90 PRINT CHR$(156); : GOTO 10
100 PRINT CHR$(158); : GOTO 10
110 PRINT CHR$(159); : GOTO 10
120 PRINT CHR$(129); : GOTO 10
130 PRINT CHR$(149); : GOTO 10
140 PRINT CHR$(150); : GOTO 10
150 PRINT CHR$(151); : GOTO 10
160 PRINT CHR$(152); : GOTO 10
170 PRINT CHR$(153); : GOTO 10
180 PRINT CHR$(154); : GOTO 10
190 PRINT CHR$(155); : GOTO 10

READY.
```

Az 1–40-es sorok beírása után a képernyő a következőképpen néz ki:

```
1 REM SZINES CSIKOK
5 PRINT CHR$(147) : REM CHR$(147)=CLR/HOME
10 PRINT CHR$(18); "           "; : REM INVERZ CSIK
20 SZ=INT(16*RND(1))+1
30 ON SZ GOTO 40,50,60,70,80,90,100,110,120,130,140,";
32 PRINT 150,160,170,180,190
40 PRINT CHR$(5); : GOTO 10

READY.
```

Szerkesztési tanácsok

Álljunk a kurzorral a 40-es sorszám 4-esére! Írjuk át 5-re, így előáll az 50-es sorszám! Most vigyük a kurzort a CHR$-függvény zárójelei közé! A SHIFT INST/DEL lenyomása után elegendő helyünk lesz egy kétjegyű szám elhelyezésére. Írjuk be a 28-ast! A RETURN billentyű lenyomása után a képernyő így néz ki:

```
1 REM SZINES CSIKOK
5 PRINT CHR$(147) : REM CHR$(147)=CLR/HOME
10 PRINT CHR$(18); "     "; : REM INVERZ CSIK
20 SZ=INT(16*RND(1))+1
30 ON SZ GOTO 40,50,60,70,80,90,100,110,120,130,140,";
32 PRINT 150,160,170,180,190
50 PRINT CHR$(28); : GOTO 10

READY.
```

Ha a programot most újra listázzuk, meggyőződhetünk arról, hogy a 40-es sor továbbra is létezik. Építsük fel hasonló módon a teljes programszöveget, majd a végső ellenőrzés után indítsuk el a programot!

Kövessük végig röviden a program működését! Az 5-ös sorban CLR/HOME CHR$-kódjának beírásával töröljük a képernyőt.

A 10-es sorban inverz kijelzési módot állítunk be, majd felrajzoljuk a színes csíkot alkotó 5 betűköz karaktert. Ez a csík az első lefutáskor világoskék.

A 20-as sor állítja elő az aktuális színt kiválasztó véletlenszámokat. A 30-as sorban az eddig még nem tárgyalt ON...GOTO (ugrás...szerint) utasítás áll, amely az IF...THEN utasítás kiterjesztésének tekinthető és többszörös programelágaztatást tesz lehetővé. Az ugrási címet az ON és GOTO között álló SZIN változó aktuális értéke alapján választja ki a számítógép a GOTO után álló sorszámok közül. Ha pl. SZIN=1, akkor az első, ha SZIN=4, akkor a negyedik címre ugrunk.

A 40–190-es sorok a 20-as sorban előállított, 1 és 16 közé eső véletlenszámot alakítják át színkóddá, majd visszairányítják a programot a 10-es sorra, és minden kezdődik elölről.

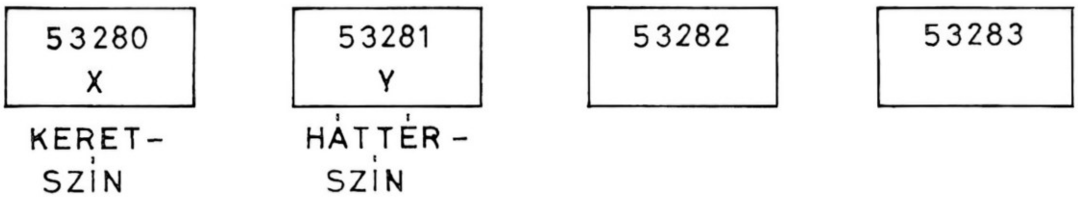
PEEK és POKE

Ebben a fejezetben egy olyan módszerrel ismerkedünk meg, amellyel "belenézhetünk" a COMMODORE 64-es belsejébe, ill. közvetlenül adatokat vihetünk a tár egy általunk választott cellájába.

Számítógépünk programtárát cellák sokaságaként képzelhetjük el, amelyekbe adatokat tölthetünk, ill. vehetünk ki. Minden egyes cella tartalmának meghatározott jelentése van. Vannak pl. olyan cellák, amelyekben a gép utánanézhet, hogy milyen

háttér- és keretszín van beállítva, ill. milyen karaktereket hol és milyen színben kell megjeleníteni a képernyőn.

A cellák tartalmának átírásával megváltoztathatjuk a felsorolt paramétereket. Módosíthatjuk pl. a színeket, alakzatokat jeleníthetünk meg és mozgathatunk a képernyőn, sőt zenei darabokat is összeállíthatunk. A programtár közvetlen írására a POKE (beszúr) utasítás szolgál, amellyel adatokat "szúrhatunk be" a kívánt cellába. A cellákat a következőképpen képzelhetjük el:



Négy tárcellát ábrázoltunk, amelyek közül kettő a képernyő keret- és háttérszínét határozza meg. Írjuk be a következőt:

```
POKE 53281,7
```

Ha megnyomjuk a RETURN billentyűt, akkor a háttérszín sárgára változik, mivel a sárga színt beállító 7-est a hátteret meghatározó cellába írtuk.

Próbálkozzunk más értékekkel is! (Bármely 0 és 255 közötti értékkel). A következő táblázat a számértékek és a színek összerendelését tartalmazza:

0	fekete	8	narancs
1	fehér	9	barna
2	piros	10	halványpiros
3	türkiz	11	sötétszürke
4	lila	12	középszürke
5	zöld	13	világoszöld
6	kék	14	világoskék
7	sárga	15	világosszürke

Nézzük végig a lehetséges háttérszín-, keretszín-kombinációkat! Ebben segít a következő rövid program:

```
10 FOR HT=0 TO 15
20 FOR KR=0 TO 15
30 POKE 53280,HT
40 POKE 53281,KR
50 FOR X=1 TO 2000 : NEXT X
60 NEXT KR : NEXT HT

READY.
```

Az összes lehetséges kombináció előállítására két egymásba skatulyázott ciklust alkalmaztunk.

Az 50-es sorba írt késleltető ciklus lelassítja a kombinációk közötti átmenetet.

Miután a program lefutott, írjuk be:

```
? PEEK(53280) AND 15
```

A válasz 15. A PEEK (benéz) utasítás kiolvassa a megadott tárcella tartamát. Az AND logikai művelettel a 15-nél nagyobb számokat "maszkoljuk ki". (Az AND utasítással később a bináris aritmetika kapcsán még részletesen foglalkozunk.) Mivel az utolsó POKE utasítással 15-öt írtunk az 53280-as keretszíncellába, a PEEK-kel a várt eredményt kapjuk.

Ha a futás közben az aktuális cellatartalmakat látni szeretnénk a képernyőn, akkor a programot a következő sorokkal kell kiegészítenünk.

```
25 PRINT CHR$(147);"KERET=";PEEK(53280)
AND 15,"HATTER="; PEEK (53281) AND 15
```

Képernyőgrafika

Számítógépünk a karaktereket mindeddig sorban egymás után írta a képernyőre, hacsak nem használtunk (,) vesszőt a PRINT utasításban, ill. nem kezdtünk új sort.

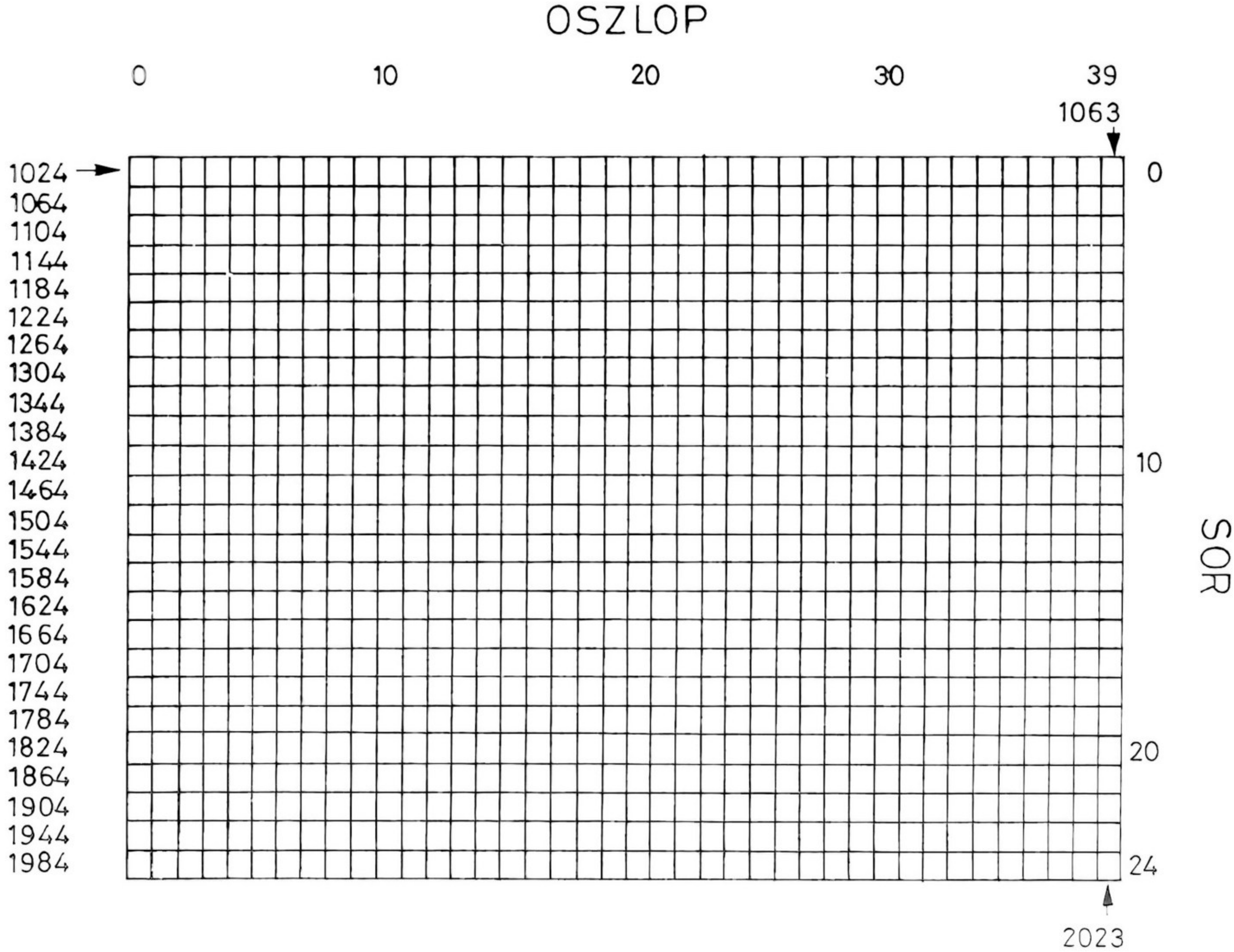
Eddig a kurzort is csak a PRINT utasítással vezéreltük. Ilyen módon a képernyő tetszőleges pontja elérhető. Ez a módszer azonban általában lassú és túlságosan sok helyet foglal el a programtárban. Ahogy azonban a tárban megvannak a színeket meghatározó cellák, ugyanúgy megtaláljuk az egyes képernyőpontokhoz tartozó cellákat is.

A képernyő tártérképe

Mivel a képernyő 25 sorból és 40 oszlopból áll, összesen 1000 karaktert írhatunk rá. Természetesen ugyanennyi helyre van szükségünk a tárban is a teljes képernyőtartalom tárolására. A képernyő olyan négyzethálóként képzelhető el, amelyben minden szem a tár egy cellájának felel meg (lásd ábra).

Ezekbe a cellákba 0 és 255 közötti értékeket írhatunk (POKE); annak, hogy az egyes számok milyen karaktereknek felelnek meg, az E Függelékben közölt táblázatban nézhetünk utána.

A COMMODORE 64-es képernyőtára az 1024-es cellával indul és a 2023-assal ér véget. Az 1024-es cella a képernyő bal felső sarkának felel meg és a tárcímek balról jobbra nőnek. A további tájékozódáshoz próbáljuk pontosan megérteni az ábrát.



Tegyük fel, hogy egy labdát szeretnénk pattogtatni a képernyőn! A labda kezdetben legyen középen: 20-as oszlop, 12-es sor. A megfelelő tárcímet a következő formulával számolhatjuk ki:

KÉPERNYŐCÍM=1024+OSZLOP+40*SOR

Esetünkben a labda kezdőcíme:

1024+20+40*12=1524

Töröljük a képernyőt SHIFT CLR/HOME -mal, majd írjuk be:

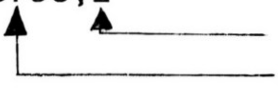
```
POKE 1524,81
```

- 81 ← Karakterkód
- 1524 ← Képernyőcím

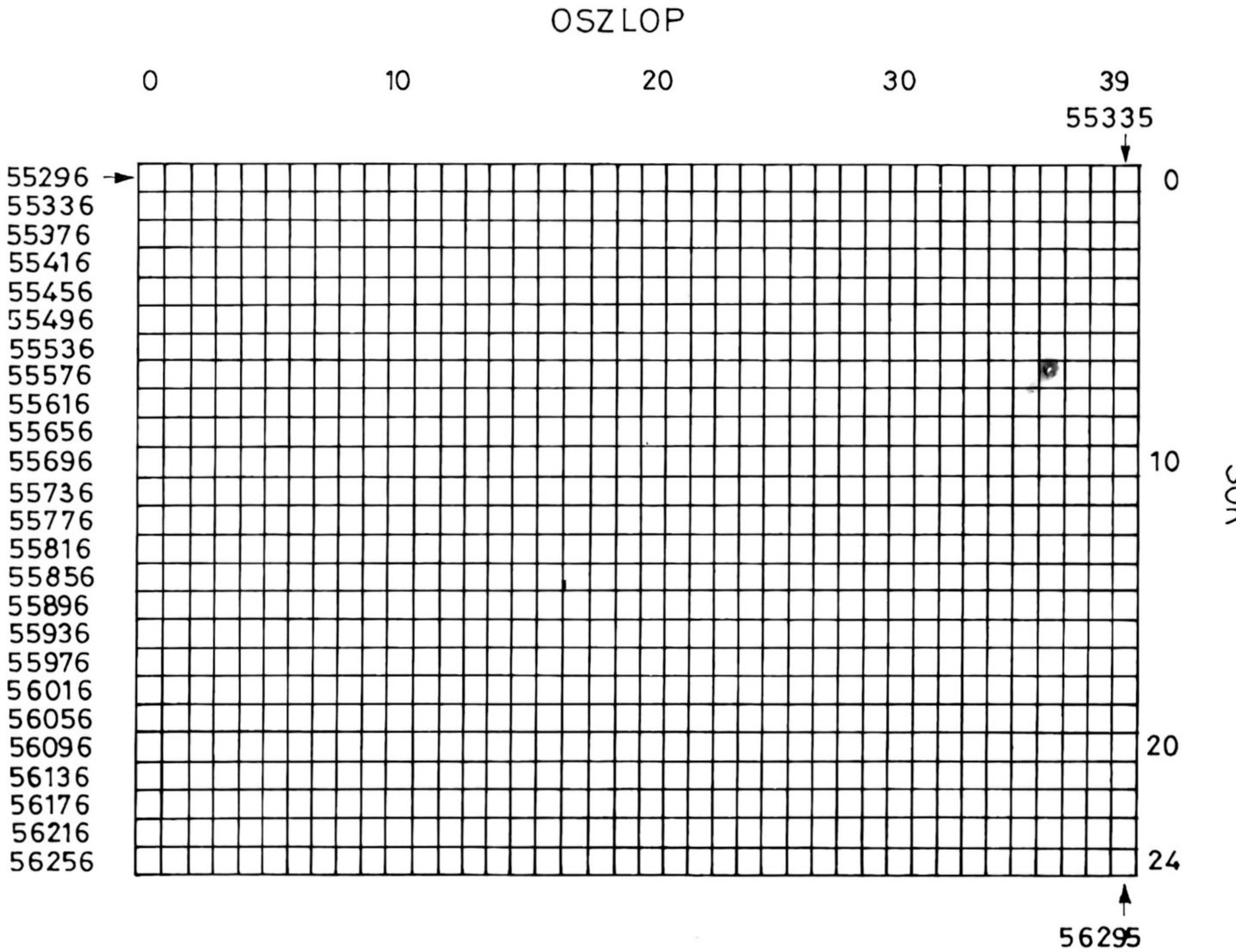
Színtár

Már van egy labdánk a képernyő közepén. Ezt nem a PRINT utasítással helyeztük el, hanem közvetlenül írtunk be egy értéket a képernyőtárba. A labdát e pillanatban azonban még nem látjuk, mivel színe megegyezik a háttér színével. A COMMODORE 64-esben a képernyőtár minden cellájához tartozik egy kiegészítő cella, amely az adott cellához tartozó színinformációt tartalmazza. Írjuk be:

```
POKE 55796,2
```



A labda ekkor piros színben jelenik meg. Összefoglalva: a képernyő minden egyes pontjához két cella tartozik a tárban, az egyik a képernyőtárban, a másik pedig a színtárban. A színtár az 55296-os cellával kezdődik (bal felső sarok), és ugyanúgy mint a képernyőtár, 1000 cellát foglal le a tárban.



A színkódok 0 és 15 közötti számok (lásd a PEEK és POKE fejezetben levő táblázatot). Az aktuális színtárcím a kezdőértéktől eltekintve ugyanúgy számítható ki, mint a képernyőcím:

```
SZÍNTÁRCÍM=55296+OSZLOP+40*SOR
```

Újabb labdajáték

Ebben a fejezetben egy a korábban bemutatottnál lényegesen jobb labdajáték-programot közlünk. A PRINT utasítás helyett a POKE-ot használjuk, amely lényegesen rugalmasabb és lehetővé teszi az összetett mozgási folyamatok ábrázolását a képernyőn.

```
10 PRINT "□" : REM CLR/HOME
20 POKE 53280,7 : POKE 53281,0
30 X=1 : Y=1
40 DX=1 : DY=1
50 POKE 1024+X+40*Y,81
55 POKE 55296+X+40*Y,1
60 FOR T=1 TO 10 : NEXT
70 POKE 1024+X+40*Y,32
80 X=X+DX
90 IF X=0 OR X=39 THEN DX=-DX
100 Y=Y+DY
110 IF Y=0 OR Y=24 THEN DY=-DY
120 GOTO 50

READY.
```

A 10-es sorban töröljük a képernyőt, a 20-asban pedig fekete hátteret és sárga keretet állítunk be.

Az X és Y változók a labda aktuális pozícióját írják le (sor és oszlop). A DX és DY változók a vízszintes és függőleges irányú elmozdulást jellemzik:

DX= 1	egy lépés jobbra
DX= –1	egy lépés balra
DX= 1	egy lépés lefelé
DX= –1	egy lépés felfelé

Az 50-es sorban felrajzoljuk a labdát a sor- és oszlopszámmal meghatározott pozícióba. A 60-as sorban a jól ismert késleltető ciklust iktattuk be, hogy a labda mozgása ne legyen élvezhetetlenül gyors.

A 70-es sorban egy betűközzel töröljük a labdát az aktuális pozícióból.

A 80-as sorban X és DX összeadásával visszük jobbra vagy balra a labdát. Ha a 90-es sorban a porgram megállapítja, hogy a labda valamelyik oldalon elérte a képernyő határát, akkor megfordítja DX előjelét.

A 100-as és 110-es sorokban ugyanez történik függőleges irányban.

A 120-as sorról visszaugrik az 50-esre, ahol a labdát az új pozícióba rajzoljuk.

Az 50-es sorban a 81-es kód megváltoztatásával a labdát tetszőleges karakterre cserélhetjük.

A következő kiegészítéssel programunkat még látványosabbá tehetjük:

```
21 FOR L=1 TO 10
25 POKE 1024+INT(RND(1)*1000),166
27 NEXT L
115 IF PEEK(1024+X+40*Y)=166 THEN DX=-DX : GOTO 80

READY.
```

A 21–27-es sorok akadályokat helyeznek el a képernyőn, véletlenszerű pozíciókba. A 115-ös sorban azt vizsgáljuk, hogy a labda nekiütközött-e egy ilyen akadálynak; ha igen, akkor DX előjelének megváltoztatásával megváltoztatjuk a mozgási irányt.

6. FEJEZET

SPRITE GRAFIKA

SPRITE-ok a COMMODORE 64-esben
A SPRITE-ok megszerkesztése
Bináris aritmetika

Bevezetés

Az előző fejezetekben azt tekintettük át, hogy hogyan jeleníthetünk meg a képernyő kívánt pontján egy karaktert a PRINT és a POKE utasítások használatával.

A mozgó grafikus alakzatok ábrázolása mindkét módszer esetében nehézségekkel jár, mivel az alakzatokat kötött szimbólumkészletből kell megszerkesztenünk, így a kialakítható formaválaszték erősen korlátozott. Ezenkívül általában a mozgás programutasításokkal való vezérlése is túlságosan bonyolult. Az ún. SPRITE-ok (szellemek) alkalmazása a legtöbb problémát megoldja. A SPRITE olyan nagyfelbontású grafikus alakzat, amelynek formája egyszerű BASIC utasításokkal írható elő. A SPRITE-okat a képernyőn egyszerűen az aktuális helyzet megadásával mozgathatjuk. Az ehhez szükséges számításokat a COMMODORE 64-es automatikusan (hardver eszközökkel) végzi el.

A SPRITE-ok színét szabadon választhatjuk meg, méretük egyetlen utasítással módosítható. Két SPRITE "elúszhat" egymás mögött képernyőn, összeütközésük pedig egyszerűen regisztrálható.

Mindezek az előnyök programozási oldalról semmiféle nehézséggel sem járnak. Mielőtt azonban hozzákezdenénk első SPRITE-unk megalkotásához, meg kell ismerkednünk a COMMODORE 64-es belső számábrázolásával. Reméljük, hogy ez a kicsi, de feltétlenül szükséges kitérő sem lesz unalmas és mindenképpen megtérül, amikor első önállóan megalkotott SPRITE-unkban gyönyörködünk.

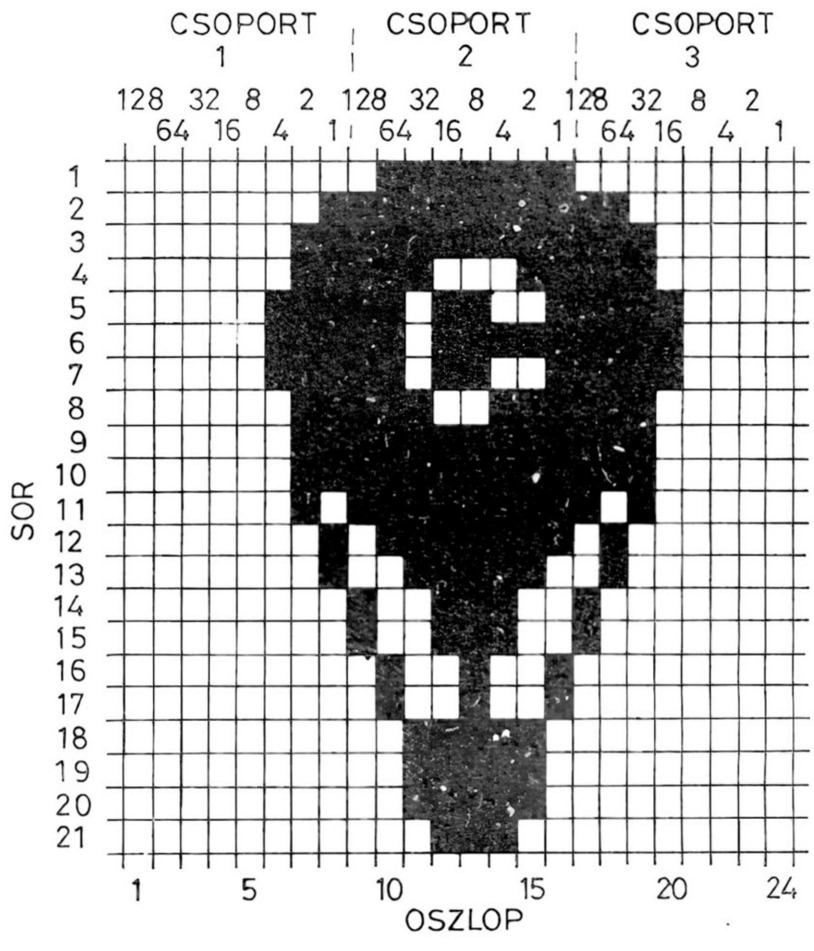
A SPRITE-ok szerkesztése

A SPRITE-ok használatát a COMMODORE 64-es egy speciális belső építőeleme a VIC (Video Interface Chip=video illesztő chip) támogatja. A munka nagy részét, amelyet a SPRITE-ok szerkesztésekor, mozgatásakor és színezésekor el kell végezni, ez a chip "vállalja magára".

A VIC a SPRITE-okra vonatkozó információ tárolására 46 regisztert használ. Ezek a regiszterek ugyanúgy épülnek fel, mint a képernyő- és a színtár cellái. Belső felépítésükről később még bővebben szólunk. A SPRITE-ok alakjára vonatkozó adatok a gép főtárában helyezkednek el. A pontos helyükre vonatkozó információt a közvetlenül a képernyőtár után következő 8 cella tartalmazza.

Ezek után nézzük meg, hogyan épülnek fel a SPRITE-ok! Amint már tudjuk, a képernyő 25 sorból és 40 oszlopból áll. Az így kialakuló 1000 képernyőcella egy négyzethálót alkot, amelynek szemeit a POKE utasítással tölthetjük fel. A valóságban minden egyes szem (karakterpozíció) egy 8x8-as pontmátrix. Így a teljes képernyő, (40x8)x(25x8)=320x200 képpontból álló terület áll rendelkezésünkre.

Példaként ebbben a 24x21-es mezőben egy léggömböt fogunk szerkeszteni. A legjobb, ha a munkát azzal kezdjük, hogy elővesszünk egy négyzetrácsos papírt és kijelölünk egy 24x21 négyzetből álló területet. Ezután rajzoljuk meg a léggömb körvonalait, majd töltsük ki azokat a négyzeteket, amelyeket a körvonal metsz! Így kialakítottuk a SPRITE formáját. A léggömb belsejét alkotó négyzeteket töltsük ki tetszésünk szerint, vagy használjuk a könyv ábráját! Most már csak az van hátra, hogy rajzunkat olyan adatsorozattá alakítsuk át, amelyet számítógépünk megért.



Osszuk fel ábránkat három egyenlő szélességű részre, amelyek mindegyike 8 oszlopból áll. Ezután a bal felső saroktól indulva írjuk rajzunk fölé háromszor egymás után a következő számsorozatot:

128,64,32,16,8,4,2,1.

A sorokat számozzuk meg 1-től 21-ig! Vegyünk fel minden sorhoz három értéktáblát az ábrán látható módon. Az első tábla mindig egy sor első 8 elemének felel meg, a második és harmadik pedig sorban, a 9-16., ill. a 17-24. elemeknek. Az értéktáblák megfelelő mezői fölé írjuk fel újra az említett 128,64,32,16,8,4,2,1 számsorozatot! (Remélhetőleg észrevettük már, hogy itt a 2 hatványairól van szó.) Ezután rendeljük rajzunk (SPRITE-unk) minden kitöltött mezőjéhez az 1-es és minden üres mezőjéhez a 0-ás számot! Ezután vegyük fel a megfelelő értékeket az egyes sorokhoz rendelt 3 értéktáblába!

Vegyük példaként a könyvben ábrázolt léggömb-SPRITE 1-es sorát! Az első 8 mező üres, így az ehhez a sorhoz tartozó első tábla csupa 0-t tartalmaz. Ahhoz, hogy a gép megértse ezt a táblát, az egyes mezők tartalmát össze kell szoroznunk az illető mező fölé írt értékkel, majd a szorzatokat össze kell adnunk. Mivel most minden mezőben 0 áll, az eredmény is 0. Az első sorhoz tartozó második tábla a következőképpen néz ki:

128	64	32	16	8	4	2	1
0	1	1	1	1	1	1	1
↑	↑	↑	↑	↑	↑	↑	↑

0 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 127

Az első sor harmadik táblája ismét csupa 0-ból áll, így az összeg is 0.

Összefoglalva, a SPRITE első sorát a 0,127,0 számsorozat jellemzi. Ezt a következőképpen fordíthatjuk le BASIC nyelvre:

DATA 0,127,0 (DATA=adat)

Nézzük most a második sort:

0	0	0	0	0	0	0	1

1 = 1

1	1	1	1	1	1	1	1
↑	↑	↑	↑	↑	↑	↑	↑

128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255

1	1	0	0	0	0	0	0
↑	↑						

128 + 64 = 192

A második sorhoz tartozó DATA utasítás:

```
DATA 1,255,192
```

A fennmaradó 19 sor "kódolását" próbáljuk meg önállóan elvégezni! Ha végeztünk, írjuk be a következő programot (a DATA és a READ (olvas) utasítással a 8. fejezetben foglalkozunk részletesebben):

```
1 REM REPULES
5 PRINT "□" : REM CLR/HOME
10 V=53248 : REM VIC KEZDOCIM
11 POKE V+21,4 : REM A 2-ES SPRITE AKTIVIZALASA
12 POKE 2042,13
14 REM A 2-ES SPRITE ADATAI A 13-AS BLOKKBOL
20 FOR N=0 TO 62 : READ Q : POKE 832+N,Q : NEXT
30 FOR X=0 TO 200
40 POKE V+4,X : REM UJ X KOOEDINATA
50 POKE V+5,X : REM UJ Y KOORDINATA
60 NEXT X
70 GOTO 30
200 DATA 0,127,0,1,255,192,3,255,224,3,231,224
210 DATA 7,217,240,7,223,240,7,217,240,3,231,224
220 DATA 3,255,224,3,255,224,2,255,160,1,127,64
230 DATA 1,62,64,0,156,128,0,156,128,0,73,0,0,73,0,0
240 DATA 62,0,0,62,0,0,62,0,0,28,0

READY.
```

A program elindítása után egy lassan szálló léggömböt látunk a képernyőn.

A program működésének megértéséhez meg kell ismerkednünk azzal, hogy a VIC egyes regiszterei pontosan mit is vezérelnek. Nézzük a következő táblázatot:

Regiszter	Leírás
0	A 0-ás SPRITE X koordinátája
1	A 0-ás SPRITE Y koordinátája
2–15	Az 1–7-es SPRITE-ok X és Y koordinátái (lásd 0–1 regiszterek)
16	Az X koordináták legmagasabb helyi értékű bitjei (bit fogalmát lásd: BINÁRIS ARITMETIKA fejezet)
21	A SPRITE-ok aktivizálása: **1 – látható; 0 – nem látható**
29	A SPRITE-ok nagyítása X irányban
23	A SPRITE-ok nagyítása Y irányban
39–46	A SPRITE-ok színei O–7

Ezenkívül a gépnek azt is tudnia kell, hogy egy adott SPRITE adatai melyik 64 cellából álló blokkban vannak. Az erre vonatkozó információt a képernyőtár után következő 8 cellában kell elhelyeznünk.

	2040	41	42	43	44	45	46	2047
	↑	↑	↑	↑	↑	↑	↑	↑
SPRITE	0	1	2	3	4	5	6	7

Végül foglaljuk össze azokat az egymást követő lépéseket, amelyeket egy SPRITE megjelenítéséhez és mozgatásához el kell végeznünk!

1. A 21-es VIC regiszterbe írjuk be a megfelelő értéket a kiválasztott SPRITE aktivizálásához!
2. A tár megfelelő cellájába helyezzük el az adott SPRITE adatait tartalmazó mutatót!
3. POKE utasításokkal írjuk a SPRITE tényleges adatait a tárba!
4. A mozgatáshoz változtassuk ciklikusan a SPRITE X és Y koordinátáit a megfelelő VIC regiszterben!
5. A SPRITE színét és nagyságát a 39–46-os, a 29-es és a 23-as VIC regiszter tartalmának megváltoztatásával módosíthatjuk.

A program néhány pontja még az eddigi magyarázatok után is homályos lehet.

A 10-es sor:

```
V=53248
```

A VIC chip kezdőcíme, azaz az első regiszter abszolút címe. Egy tetszőleges VIC regiszter abszolút címét úgy kaphatjuk meg, hogy a regiszter sorszámát hozzáadjuk a kezdőcímhez.

A 11-es sor:

```
POKE V+21,4
```

Ez az utasítás láthatóvá teszi a 2-es SPRITE-ot. Ennek oka, hogy a POKE utasítás hatására a 21-es VIC regiszter 2-es mezőjébe 1 kerül, ha a regiszterbe 4-et írunk (lásd BINÁRIS ARTIMETIKA fejezet).

	128	64	32	16	8	4	2	1	— a mezőhöz rendelt érték
	7	6	5	4	3	2	1	0	— a SPRITE sorszáma
21	0	0	0	0	0	1	0	0	=4

↖ 1-es a SPRITE aktivizálásához

Az összes SPRITE aktuális állapota (látható, nem látható) a 21-es regiszter tartalmától függ. Ha pl. a 3-as SPRITE-ot szeretnénk láthatóvá tenni, akkor a regiszterbe 8-at kell írnunk. Ha mind a 2-es, mind a 3-as SPRITE-ot aktivizálni akarjuk, akkor a regiszterbe 12-t (8+4=12) kell helyeznünk. Az ehhez szükséges POKE utasítás:

```
POKE V+21, 12
```

lenne.

A 12-es sor:

```
POKE 2042, 13
```

Itt arra utasítjuk a gépet, hogy a 2-es SPRITE adatait a 13-as blokkból olvassa ki. Egy SPRITE adatai 63 cellát foglalnak le a tárban. Egy értéktábla tartalma 1 cellának felel meg. Minden sorhoz 3 értéktábla, azaz 3 cella szükséges, az összes igénybe vett tárterület 21 sor x 3 cella = 63 cella. Egy SPRITE megjelenítéséhez tehát 63 tárcella tartalmát kell beírnunk a következő módon:

```
20 FOR N=0 TO 62: READ Q: POKE 832+N,Q: NEXT
```

Ez a ciklus 63 db cellát tölt fel adattal a 13-as blokkban (1 blokk=64 cella). A 13-as cella a 13x64=832-es címen kezdődik a tárban.

```
30 FOR X = 0 TO 200
40 POKE V+4,X          A 2-es SPRITE X koordinátája
50 POKE V+5,X          A 2-es SPRITE Y koordinátája
60 NEXT X
```

Mivel a 4-es és 5-ös VIC regiszter a 2-es SPRITE X és Y koordinátáját tartalmazza ez a programszakasz a SPRITE átlós irányú mozgatását hajtja végre a képernyőn. A koordináta-rendszer középpontja a bal felső sarok, így a mozgást kijelölő szakasz kezdőpontja a képernyő bal felső, végpontja pedig a jobb alsó sarka. Mivel a léggömb újrarajzolása az új pozícióban megfelelően gyors, a mozgás folytonosnak tűnik (erről a témáról bővebben lásd az O Függeléket).

Ha egyszerre több SPRITE-ot is meg akarunk jeleníteni, akkor mindegyikhez saját tárterületet kell hozzárendelnünk.

A 70-es sorból a program visszaugrik a 30-asra, és az egész repülés újra kezdődik. A program maradék részét azok a DATA utasítások töltik ki, amelyek a léggömb alakját leíró adatokat tartalmazzák. Szúrjuk be a következő sort, majd indítsuk újra a programot:

```
25 POKE V+23,4: POKE V+29,4: REM NAGYITAS
```

A léggömb most kétszer akkora, mint eddig. Ezt úgy értük el, hogy a 23-as és a 29-es regiszterbe 4-et (mivel a 2-es SPRITE-ról van szó) írtunk. Jegyezzük meg jól, hogy a SPRITE bal felső sarka a nagyítás után helyben marad. Programunkat a követkző sorok beszúrásával még érdekesebbé tehetjük:

```
11 POKE V+21,12
12 POKE 2042,13 : POKE 2043,13
30 FOR X=1 TO 190
45 POKE V+6,X
55 POKE V+7,190-X

READY.
```

Most egy második léggömb is megjelenik a képernyőn, mivel a 21-es VIC regiszterbe 12-t írtunk. Ez azt jelenti, hogy a regiszter 2. és 3. mezőjébe is 1 kerül, azaz mind a 2-es, mind a 3-as SPRITE-ot aktivizáljuk.

A 45–55-ös sorok a 3-as SPRITE mozgását vezérlik az X és Y koordinátákat tartalmazó 6-os és 7-es VIC regiszterek tartalmának változtatásával. Ha még mindig nem lennénk elégedettek a látvánnyal, akkor bővítsük tovább a programot:

```
11 POKE V+21,28
12 POKE 2042,13 : POKE 2043,13 : POKE 2044,13
25 POKE V+23,12 : POKE V+29,12
48 POKE V+8,X
58 POKE V+9,100

READY.
```

A 11-es sor POKE parancsa az eddigiek mellett a 4-es SPRITE-ot is aktivizálja, mivel hatására a 21-es VIC regiszter 4-es mezőjébe is 1 kerül (4+8+16=28). A 12-es sor azt közli a géppel, hogy mindhárom SPRITE adatait a 13-as blokkból vegye.

A 25-ös sorban nagyítjuk a 2-es és a 3-as SPRITE-ot (4+8=12). A 48-as és 50-es sorok a 4-es SPRITE-ot vízszintes irányban mozgatják a képernyőn.

Néhány megjegyzés a SPRITE-okhoz

A már megismert színkódolási módszerrel (lásd 5. fejezet vagy G Függelék) az egyes SPRITE-okat 16 különböző színben jeleníthetjük meg.

Ha pl. az 1-es SPRITE-nak a zöld színt szánjuk, akkor a következő POKE utasítással érhetjük el a célunkat:

```
POKE V+40,13 (ahol V=53248)
```

Talán észrevettük már, hogy miközben a példaprogramot futtatjuk, a SPRITE-ok soha nem érik el a képernyő jobb oldali határát. Ennek oka, hogy a VIC X regisztereibe írható szám maximális értéke 255, a teljes képernyő pedig 320 képpont szélességű.

Ha a mozgásokhoz a teljes képernyőt szeretnénk kihasználni, akkor a 16-os VIC regiszterrel is foglalkoznunk kell. Ha ennek a regiszternek az n-es mezője 1-et tartalmaz, akkor az n-es SPRITE X koordinátája nagyobb, mint 255. A pontos érték ekkor:

X regiszter tartalma + 256.

Ha a 2-es SPRITE X koordinátáját pl. 256 és 320 között akarjuk felvenni, akkor 16-os VIC regiszterbe 4-et kell írnunk, hogy a 2-es cellába 1 kerüljön:

```
POKE V+16,4
```

Ha most a 2-es SPRITE-hoz tartozó X regiszter (4-es VIC regiszter) tartalmát 0 és 63 között változtatjuk, akkor a teljes X koordináta érték 256+0=256 és 256+63=319 között mozog. A most elmondottakat úgy érthetjük meg a legkönnyebben ha elemezzük a következő programot, amely az eddig használt SPRITE programok egy újabb változata.

```
5 PRINT "□" : REM CLR/HOME
10 V=53248 : POKE V+21,4 : POKE 2042,13
20 FOR N=0 TO 62 : READ Q : POKE 832+N,Q : NEXT
25 POKE V+5,100
30 FOR X=0 TO 255
40 POKE V+4,X
50 NEXT
60 POKE V+16,4
70 FOR X=0 TO 63
80 POKE V+4,X
90 NEXT
100 POKE V+16,0
110 GOTO 30
200 DATA 0,127,0,1,255,192,3,255,224,3,231,224
210 DATA 7,217,240,7,223,240,7,217,240,3,231,224
220 DATA 3,255,224,3,255,224,2,255,160,1,127,64
230 DATA 1,62,64,0,156,128,0,156,128,0,73,0,0,73,0,0
240 DATA 62,0,0,62,0,0,62,0,0,28,0

READY.
```

A 60-as sorban a 2-es SPRITE X koordinátájának legmagasabb helyi értékű bitjét állítjuk be. A 70-es sorban kezdődik az a programrész, amely a léggömböt a képernyő jobb oldali határáig viszi. A 100-as sor is fontos, mivel itt állítjuk 0-ra az X koordináta legmagasabb helyi értékű bitjét, így a léggömb újra a képernyő bal oldali határáról indulhat.

Bináris aritmetika

Nem halaszthatjuk tovább, hogy megismerkedjünk az adatok gépi ábrázolásmódjával, ill. a feldolgozásukhoz használt bináris aritmetikával. Nézzük először az alapfogalmakat:

BIT: Az információ legkisebb egysége, amelyet egy számítógép kezelni tud. Egy bit két különböző értéket (vagy állapotot) vehet fel, akár egy kétállású kapcsoló. A "bekapcsolt" állapot az 1-es, a "kikapcsolt" pedig a 0-ás.

BYTE: 8 egymás melletti bitből épülő információegység. A bitek értékétől függően 256 különböző bitkombináció képzelhető el. Így egy byte értéke 0 és 255 között változhat.

1. Példa: A byte minden bitje 0

128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	0

A byte értéke:

0*128+0*64+0*32+0*16+0*8+0*4+0*2+0*1=0
2. példa: A byte minden bitje 1:

128	64	32	16	8	4	2	1
1	1	1	1	1	1	1	1

A byte értéke:

1*128+1*64+1*32+1*16+1*8+1*4+1*2+1*1=255

A regiszter

speciális tárcella a számítógépben. A COMMODORE 64-esben egy regiszterben 1 byte, azaz egy 0 és 255 közötti szám tárolható.

A megfelelő regiszterek tartalmának változtatásával vezérelhetjük a gép műveletvégzését. A SPRITE-ok tárgyalásakor pl. olyan regszterekkel ismerkedtünk meg, amelyek az alakzatok mozgásáért, láthatóvá tételéért, nagyításáért stb. felelősek.

Bináris szám decimálissá alakítása

A következő táblázat a 2-es szám első 8 hatványát ábrázolja bináris alakban:

DECIMÁLIS SZÁM								
128	64	32	16	8	4	2	1	
0	0	0	0	0	0	0	1	2↑0
0	0	0	0	0	0	1	0	2↑1
0	0	0	0	0	1	0	0	2↑2
0	0	0	0	1	0	0	0	2↑3
0	0	0	1	0	0	0	0	2↑4
0	0	1	0	0	0	0	0	2↑5
0	1	0	0	0	0	0	0	2↑6
1	0	0	0	0	0	0	0	2↑7

Bizonyára kitaláltuk már, hogy a POKE utasítással a tárba írható decimális számnak azért kell 0 és 255 közé esnie, mivel a tár egy adott cellája 1 byte-ot (8 bitet) tárolhat. Ha tudni szeretnénk, hogy egy adott bináris szám melyik decimális számnak felel meg, akkor csak össze kell adni a 2-es szám azon hatványait, amelyeknek helyén a byte bitmintájában 1-es áll. Nézzük pl. a 01010011 bináris szám decimálissá alakítását:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	0	0	1	1

$$2 + 1 = 3$$

A következő segédprogram a beadott bináris számot decimálisssá alakítja.

```
5 REM BINARIS-DECIMALIS ATVALTAS
10 INPUT "A 8 BITES BINARIS SZAM: "; A$
12 IF LEN(A$)<>8 THEN PRINT "8 BITET KEREK..." : GOTO 10
15 DE=0 : C=0
20 FOR X=8 TO 1 STEP -1 : C=C+1
30 DE=DE+VAL(MID$(A$,C,1))*2↑(X-1)
40 NEXT X
45 PRINT
50 PRINT "BINARIS "; A$;" = "; "DECIMALIS "; DE
55 PRINT
60 GOTO 10

READY.
```

A beírt bináris számot az A$ karakterfüzér változóban tároljuk. A karakterfüzér elemeit (ebben az esetben az egymást követő biteket) a MID$ függvénnyel olvassuk ki jobbról balra haladva. A VAL függvénnyel megvizsgáljuk, hogy az éppen soron levő elem 0 vagy 1, az így meghatározott értéket pedig megszorozzuk a 2-es szám megfelelő hatványával. Az így kapott szorzatokat aztán összegezzük, majd az 50-es sorban kiíratjuk mind a bevitt bináris, mind az eredményül kapott decimális számot.

7. FEJEZET

HANGKELTÉS A COMMODORE 64-ESEN

A hangkeltő programok szerkezete
Zenélő program
Hangjellemzők beállítása
Hanghatárok

A számítógépes hangkeltésnek (gyakran hanggenerálásnak mondjuk) két fő alkalmazási területe a speciális hanghatások létrehozása, ill. az egyszerű zenei darabok lejátszása. Ezért röviden foglalkozunk a zeneprogramok szerkezetével, majd bemutatunk egy olyan programot, amellyel szabadon kísérletezhetünk.

A hangkeltő programok szerkezete

Egy hang hangzását négy jellemző határozza meg: a hangmagasság, a hangerő, a hangszín és a leütés módja. Tulajdonképpen az utolsó két tulajdonság az, amelynek alapján a hangszerek hangját meg tudjuk különböztetni egymástól.

A COMMODORE 64-esben a hanggenerálást egy beépített alkatrész, a SID (Sound Interface Device=Audio Illesztő Készülék) végzi. A SID belső regisztereinek megfelelő feltöltésével érhetjük el a kívánt hangzást.

A COMMODORE 64-es három szólamot tud megszólaltani, egyelőre azonban foglalkozzunk csak az elsővel.

A SID kezdőcíme, amelyet az SI változóban tárolunk 54272

```
SI=54272
```

A hangmagasságot fizikailag a hang frekvenciája határozza meg. A frekvencia-paramétert – amely 0 és 65000 közötti szám lehet – a SID az első két regiszterében tárolja. Azért van szükség két regiszterre, mert ilyen nagy számokat csak két byte-on tudunk ábrázolni. A frekvenciaérték alacsonyabb helyi értékű byte-ját Lo-Byte-nak (low=alacsony), a magasabb helyi értékűt pedig Hi-Byte-nak (high=magas) nevezzük.

```
FL=SI    (Frekvencia, Lo-Byte)
FH=SI+1  (Frekvencia, Hi-Byte)
```

A hangerőt 16 fokozatban szabályozhatjuk. A hangerő paraméter 0 (kikapcsolt állapot) és 15 (teljes hangerő) közötti szám a 24-es SID regiszterben:

```
HE=SI+24       (Hangerő)
```

A hangszínt lényegében a hang hullámformája határozza meg. A COMMODORE 64-es négy alapformát kínál: háromszög, fűrészfog, négyszög és zaj. Szakkönyvekből megtanulhatjuk, hogy milyen módon befolyásolhatjuk ezeket az alapformákat pl. szűrők beiktatásával. Most azonban elégedjünk meg az alaplehetőségekkel! A hullámformákat egy-egy bit vezérli a 4-es SID regiszterben:

```
HF=SI+4        (Hullámforma)
```

A megadott hullámformák kiválsztásához rendre 17-et, 33-at, 65-öt vagy 129-et kell írnunk ebbe a regiszterbe. Ha a négyszögjel (azaz 65) mellett döntünk, akkor

még egy kiegészítő paramétert is meg kell adnunk, mégpedig a kitöltési tényezőt. A kitöltési tényező szabja meg a négyszögjel "be-" és "kikapcsolt" állapotának egymáshoz való viszonyát. Értéke 0 és 4095 között változhat.

KL=SI+2	(Kitöltés, Lo-Byte)
KH=SI+3	(Kitöltés, Hi-Byte)

A legérdekesebb azonban még csak most következik: A COMMODORE 64-es lehetőséget ad az egyes hangok burkológörbéjének meghatározására. Nézzük meg kicsit részletesebben, hogy mit is jelent ez pontosan! Egy hang megszólalása és elhallgatása közötti időtartam négy jól elkülöníthető szakaszra bontható: felfutás, csillapodás, tartás és kioltás (lecsengés). Amikor a COMMODORE 64-es megszólaltat egy hangot, akkor az először fejfut a maximális hangerőre, majd fokozatosan csillapodik a tartási paraméterrel meghatározott hangerőre. Ilyen hangosan szól egészen addig, ameddig a hangot ki nem kapcsoljuk, ezután fokozatosan cseng. A burkológörbét meghatározó négy paraméter számára két regiszter áll rendelkezésre. A felfutási és csillapodási paramétert a "leütés" regiszterben, a tartási és kioltási paramétert pedig a "tartós" regiszterben tároljuk. Az egyes paraméterek négy-négy bitet foglalnak el:

L =SI+5	(Leütés) felfutás: felső négy bit csillapodás: alsó négy bit
T =SI+6	(Tartás) tartás: felső négy bit csengés: alsó négy bit

Ha a felfutási és lecsillapodási paramétert kicsire választjuk, akkor a hang gyorsan és keményen csendül fel, míg nagy érték esetén lassan és lágyan. A T (tartás) regiszter felső négy bitjébe írt maximális érték a 24-es HE (hangerő) regiszterben beállított hangerőt eredményezi tartási szintként, a kisebb értékek pedig csökkentik ezt a szintet. A lecsengésre ugyanaz igaz, mint a felfutásra: kis érték esetén keményen, nagy érték esetén pedig lassan és lágyan lecsengő hangot kapunk.

Példaprogram

Először is azt kell eldöntenünk, hogy melyik szólamot (vagy szólamokat) szeretnénk megszólaltatni. Ha minden szólamhoz beállítjuk a megfelelő paramétereket, akkor egyidejűleg akár mindhármat is hallhatjuk. A példában az egyszerűség kedvéért csak az 1-es szólamot használjuk.

`10 SI=54272:FL=SI:` `FH=SI+1:HF=SI+4:` `L=SI+5:T=SI+6:` `HE=SI+24`	1. A regisztercímek definiálása
`20 POKE HE,15`	2. Maximális hangerő
`30 POKE L,16+9`	3. Leütés
`40 POKE T,4*16+4`	4. Tartás és lecsengés
`50 POKE FH,29:POKE FL,69`	5. A frekvencia magasabb és alacsonyabb helyi értékű byte-ja a normál zenei A hanghoz. A többi hanghoz tartozó értékeket a P Függelék táblázatából olvashatjuk ki
`60 POKE HF,17`	6. Hullámforma. Mindig utolsóként kell beállítani, mivel a HF regiszter legkisebb helyi értékű bitje kapcsolja be, ill. ki a hanggenerátort. Ha ez a bit 1, akkor a hang megszólal, ha 0, akkor pedig elhallgat
`70 FOR HH=1TO500:NEXT`	7. Ciklus a hang hosszának szabályozásához
`80 POKE W,0:POKE L,0:` `POKE T,0`	8. A hanggenerátor kikapcsolása és burkoló görbe beállítása

A RUN parancs beírása után felcsendül a program által keltett hang.

Zene a COMMODORE 64-esen

Nem kell zeneművésznek lennünk ahhoz, hogy a COMMODORE 64-esen egyszerű dallamokat szólaltassunk meg. Írjuk be a következő példaprogramot, amely a rendelkezésre álló három szólamból csak egyet használ ki:

`10 REM HANGSKALA`	Programnév
`20 SI=54272:FL=SI:FH=SI+1:` `HF=SI+4:L=SI+5:T=SI+6:` `HE=SI+24`	Regisztercímek definiálása
`30 POKE HE,15`	Maximális hangerő
`40 POKE L,9`	Leütés
`50 READ X:READ Y`	A frekvencia magasabb és alacsonyabb helyi értékű byte-jának kiolvasása DATA sorokból

`60 IFY=-1THEN POKE HF,0:END`	Ha a program –1-et talál, akkor kikap csolja a hanggenerátort, majd leáll
`70 POKE FH,X;POKE FL,Y`	A két frekvenciaregiszter feltöltése POKE-kal
`80 POKE HF,17`	Hullámforma beállítása és a hang generátor bekapcsolása
`90 FOR HH=1TO100:NEXT`	Hangzáshossza
`100 POKE HF,0`	Hanggenerátor kikapcsolása
`110 FOR HH=1TO50:NEXT`	Rövid szünet a lecsengés után
`120 GOTO40`	Következő hang
`130 DATA 17,103,19,137,21,237, 23,59,26,20,29,69,32, 219,34,207`	Ezek a számpárok a C-dur skála hangjainak frekvenciaértékeit adják, váltakozva a magasabb és alacsonyabb helyi értékű byte-okat
`140 DATA-1,-1`	Ezek a frekvenciaként értelmetlen értékek jelzik a programnak a 60-as sorban, hogy a hangskála végéhez ért

Ha egy csembaló hangzását szeretnénk utánozni, akkor a 80-as sort így kell módosítanunk:

```
80 POKE HF,33
```

Ezzel a POKE utasítással fűrészfog alakú hullámformát állítunk be az eddig használt háromszög helyett, így élesebb, felhangokban gazdagabb hangzást érünk el. Azonban a hullámforma csak egy a rendelkezésre álló számos lehetőségből a hangzás meghatározásához. A megfelelő leütésérték (felfutás és csillapodás) kiválasztásával csembalónkat könnyen bendzsóvá "változtathatjuk". Változtassuk meg a 40-es sort:

```
40 POKE A,3
```

Ezzel a módszerrel számos hangszer hangzását utánozhatjuk, azaz a COMMODORE 64-est szintetizátorként működtethetjük. A pontos tennivalókat a következő szakaszban foglaltuk össze.

A hangjellemzők beállítása

1. Hangerő – A beállított érték mindhárom szólamra érvényes. A kérdéses regiszter címe 54296. A maximális hangerőt akkor kapjuk, ha 15-öt írunk a regiszterbe:

```
POKE SI+24,15 vagy POKE 54296,15
```

A 0 érték kikapcsolja a hanggenerátort:

```
POKE SI+24,15 vagy POKE 54296,0
```

A hangerőt általában a zeneprogramok elején, egyszer állítjuk be; programozott változtatásával azonban érdekes hanghatásokat érhetünk el.

2. Hullámforma – Amint a példákból is kiderült, erőteljesen hat a hangzásra. Az egyes szólamokhoz külön-külön állítható be. Négy lehetőség közül választhatunk; háromszög, fűrészfog, négyszög és zaj.

A következő táblázat összefoglalja a hullámformák kiválasztására vonatkozó tudnivalókat:

A hullámformák beállítása

	REGISZTER			TARTALOM			
SZÓLAM	1	2	3	ZAJ	NÉGYSZÖG	FŰRÉSZ	HÁROMSZÖG
	4	11	18	129	65	33	17

Ha pl. az 1-es szólamhoz a háromszöghullámot akarjuk választani, akkor a következő POKE utasítást kell végrehajtanunk.

```
POKE SI+4,17 vagy POKE 54276,17
```

Ezt a táblázatot használtuk, amikor a hangskálaprogramban a csembalós hangzás eléréséhez a 4-es regiszter tartalmát 33-ra módosítottuk.

3. Burkológörbe – Az egyes szólamokhoz külön-külön beállítható. A felfutási paraméter a teljes hangerő elérésének, a csillapodási paraméter pedig a tartási szintre való visszaesésnek az idejét adja meg. Ha a tartási szintet 0-nak választjuk, akkor a csillapodási paraméter határozza meg a hang hosszát. A leütésregiszterbe a felfutás és a lecsillapodás megfelelően összegzett értékét kell írnunk (lásd bináris aritmetika). A címeket és az összegzés módját a következő táblázatból olvashatjuk ki:

A leütés beállítása

	REGISZTER			TARTALOM	
SZÓLAM	1	2	3	FELFUTÁS	LECSILLAPODÁS
	5	12	19	15*16(lágy)....0*16(kemény)	15(lágy)...0(kemény)

A felfutás és a csillapodás viszonyát úgy ismerhetjük fel a legegyszerűbben, ha pl.

```
POKE SI+5,66             (SI+5=54277)
```

helyett a

```
POKE SI+5, 4*16+2
```

formát használjuk.

Ebből világosan kiderül, hogy közepes értékű felfutást és kis értékű lecsillapodást adtunk meg.

A hangbeállításról eddig elmondottakat egy példaprogramban foglaltuk össze:

```
 10 REM GYAKORLOPROGRAM
 20 SI=54272:FL=SI:FH=SI+1:KL=SI+2:
    KH=SI+3:HF=SI+4:L=SI+5:T=SI+6:
    HE=SI+24
 30 PRINT"NYOMJON LE EGY BILLENTYUT!"    Üzenet a képernyőre
 40 GETZ$:IFZ$=""THEN40                  Történt-e billentyűleütés
 50 POKE HE,15                           Hangerő
 60 POKE L,1*16+5                        Felfutás és csillapodás
 70 POKE T,0*16+0                        Tartás és lecsengés
 80 POKE KH,8:POKE KL,0                  Kitöltési tényező
 90 POKE FH,14:POKE FL,162               Frekvencia
100 POKE HF,17                           Hullámforma, hanggenerá
                                         tor bekapcsolása
110 FORHH=1TO200:NEXT                    Hangzáshossza
120 POKE HF,0                            Hanggenerátor leállítása
130 GOTO40                               Ugrás a program elejére
```

A program egy 1-es szólamban megszólaló rövid felfutású, közepes csillapodású hangot generál, amely egy bádogdobozban pattogó labdát idéz.

A 60-as sor módosításával tetszésünk szerint módosíthatjuk a hanghatást:

```
60 POKE L, 11*16+14
```

A most hallott hang leginkább valamilyen fúvós hangszerre, pl. egy oboára

emlékeztet. Próbáljunk ki még néhány hullámformát és burkológörbét, hogy magunk is érezzük a paramétereknek a hang karakterére gyakorolt hatását.

A tartási paraméterrel azt a hangerőt írjuk elő, amelyre a hang a felfutás után visszaesik. A hang hosszát ekkor a szokásos módon egy FOR...NEXT ciklus szabályozza. A tartás-lecsengés regiszterbe a két paraméter megfelelően összegzett értékét kell írnunk (lásd bináris aritmetika). A címeket és az összegzés módját a következő táblázatból olvashatjuk ki:

Tartás/lecsengés

	REGISZTER			TARTALOM	
SZOLAM	1	2	3	TARTÁS	LECSENGÉS
	6	13	20	15*16(hangos)..0*16(néma)	15(lassú)...0(gyors)

Cseréljük ki a 70-es sorban a nullákat valamilyen más értékre (a maximum 15) és hallgassuk meg, hogy mi lesz belőle.

4. A szólam és a hangjegyek kiválasztása – A hangmagasság meghatározásához két paramétert, a frekvencia magasabb és alacsonyabb helyi értékű byteját kell beállítanunk. A hangskála hangjegyeinek frekvenciaértékeit a P Függelék táblázata tartalmazza. Mivel a frekvenciaértékeket minden szólamhoz külön-külön adhatjuk meg, a COMMODORE 64-est úgy is beprogramozhatjuk, hogy egy dallamot három szólamban játsszon le.

Frekvenciabeállítás az 5. oktávban

	REGISZTER			POKE ÉRTÉKEK AZ 5. OKTÁVBAN												
SZÓLAM	1	2	3	C	C#	D	D#	E	F	F#	G	G#	A	A#	H	C
HI-BYTE	1	8	15	35	37	39	41	44	46	49	52	55	58	62	66	70
LO-BYTE	0	7	14	3	24	77	163	29	188	132	117	148	226	98	24	6

Az 5. oktáv C hangjának megszólaltatásához az 1-es szólamban a következő POKE utasításokat kell végrehajtanunk:

```
POKE SI+1,35:POKE SI,3
```

vagy

```
POKE 54273,35:POKE 54272,3
```

Ugyanez a 2-es szólamban:

```
POKE SI+8,35:POKE SI+7,3
```

vagy

```
POKE 54280,35:POKE 54279,3
```

Komponálás a COMMODORE 64-esen

A következő példaprogrammal dalokat komponálhatunk és játszhatunk le az 1-es szólamban. Figyeljük meg, hogy a 20-as sorban a SID regiszterek címeit numerikus változókhoz rendeltük, így a program lényegesen áttekinthetőbbé vált!

Tanulmányozzuk gondosan a DATA utasítások használatát is! Az egyes hangok leírásához szükséges paramétereket, a frekvencia magasabb és alacsonyabb helyi értékű byte-ját, valamint a hang hosszát hármas egységekben, adatként tároltunk.

A hangok hosszát egy FOR. . .NEXT ciklus szabályozza, amelyhez a cilusváltozó végértékét (azaz a hang megszólalásának időtartamát) a DATA sorokból olvassa ki a program. Az egyes hangértékeknek megfelelő számok:

nyolcad: 125
negyed: 250
pontozott negyed: 375
fél: 500
egész: 1000

Ezeket az ütemadatokat természetesen izlésünk szerint módosíthatjuk. Nézzük most a 110-es sort: a 17 és a 103 a C hang frekvenciájának magasabb és alacsonyabb helyi értékű byte-ja, a 250 pedig azt jelzi, hogy az első hang egy negyed. Hasonlóan gondolkozva megállapíthatjuk, hogy a második hang egy negyed hosszúságú E.

Saját dalunkat a P Függelék hangjegytáblázatának felhasználásával komponálhatjuk meg. A dal hosszának csak a COMMODORE 64-es tárkapacitása szab határt. Csupán arról kell gondoskodnunk, hogy az utolsó három DATA érték –1 legyen, mivel ez jelzi a programnak, hogy a dal végéhez ért.

```
10 REM KOMPONALAS
20 SI=54272 : FL=SI : FH=SI+1 : HF=SI+4 : L=SI+5";
22 PRINT T=SI+6 : HE=SI+24
25 KL=SI+2 : KH=SI+3
30 POKE HE,15 : POKE KH,13 : POKE KL,15";
32 POKE L,3*16+15 : POKE T,9
40 READ X : READ Y : READ HH
50 IF X=-1 THEN END
60 POKE FH,X : POKE FL,Y
70 POKE HF,65
80 FOR S=1 TO HH : NEXT
90 POKE HF,0
100 GOTO 40
110 DATA 17,103,250,21,237,250,26,20,400,21
```

```
120 DATA 237,100,26,20,250,29,69,250
130 DATA 26,20,250,0,0,250,21,237,250,26,20,250,29
140 DATA 69,1000,26,20,250,0,0,250
150 DATA -1,-1,0

READY.
```

Hanghatások

A hanghatásokat általában arra használjuk, hogy felhívjuk velük a figyelmet valamely programban lejátszódott lényeges eseményre. Így jelezhetjük pl., hogy a felhasználó éppen saját adatlemezét próbálja törölni, vagy hogy egy játékprogramban éppen felrobbant egy találatot kapott hajó stb.

Néhány javaslat a kísérletezgetéshez:

1. Ha egy éppen megszólaltatott hang hangerejét megváltoztatjuk, akkor visszhanghatás jön létre.
2. Ha gyorsan ugrálunk két eltérő frekvenciájú hang között, akkor ezzel tremolo hatást kelthetünk.
3. Próbáljuk ki a különböző hullámformák által kínált lehetőségeket!
4. Ne sajnáljuk az időt a burkológörbe beállítására és változtatására!
5. Több szólam egyidejű megszólaltatása eredeti hanghatásokat eredményezhet.
6. Ne féljünk a négyszöghullámtól, próbáljuk ki a különféle pulzushosszakat!
7. Kísérletezzünk a zajgenerátorral, próbáljuk utánozni a robbanás, taps, lövés, lépések stb. hangját!
8. Változtassuk gyors egymásutánban a frekvenciát több oktávon keresztül!

Példák hanghatásokra

Az itt megadott programrészleteket változatlan vagy tetszésünk szerint módosított formában építhetjük be egy BASIC nyelvű programba. Néhány további programozási ötlettel támogatjuk a COMMODORE 64-es hanggenerálási lehetőségeinek teljes kihasználását.

```
0 REM BABA
0 SI=54272 : FL=SI : FH=SI+1 : HF=SI+4 : L=SI+5";
2 T=SI+6 : HE=SI+24
5 KL=SI+2 : KH=SI+3
0 POKE HE,15 : POKE KH,15 : POKE KL,15";
2 POKE L,0*16+0 : POKE T,15*16
0 POKE HF,65
0 FOR X=250 TO 0 STEP -2 : POKE FH,40";
2 POKE FL,X : NEXT
0 FOR X=150 TO 0 STEP -4 : POKE FH,40";
2 POKE FL,X : NEXT
0 POKE HF,0

EADY.
```

```
10 REM LOVES
20 SI=54272 : FL=SI : FH=SI+1 : HF=SI+4 : L=SI+5
22 T=SI+6 : HE=SI+24
25 KL=SI+2 : KH=SI+3
30 FOR X=15 TO 0 STEP -1
40 POKE HE,X : POKE L,15 : POKE T,0 : POKE FH,40
45 POKE FL,200 : POKEHF,129
50 NEXT
60 POKE HF,0 : POKE L,0

READY.
```

```
10 REM MOTOR
20 SI=54272
30 FOR K=0 TO 24 : READ X : POKE SI+K,X : NEXT
40 DATA 9,2,0,3,0,0,240
50 DATA 12,2,0,4,0,0,192
60 DATA 16,2,0,6,0,0,64
70 DATA 0,30,243,31 : REM SZURO
80 POKE SI+4,65 : POKE SI+11,65 : POKE SI+18,65

READY.
```

8. FEJEZET

PROGRAMOZÁS HALADÓKNAK

READ és DATA utasítások
Középérték-számítás
Indexelt változók
Egydimenziós tömbök
Dimenzionálás
Kockajáték
Kétdimenziós tömbök

READ és DATA

Az eddigiekben két olyan módszerrel ismerkedtünk meg, amelyekkel egy változónak értéket adhatunk: a programon belül közvetlenül (pl. A=2), ill. az INPUT utsítással.

Sokszor azonban – különösen nagyobb mennyiségű adat feldolgozásakor –, egyik módszer sem igazán célravezető. Ilyen esetekben – ahogy azt a most következő program is példázza – legokosabb a következő módon eljárni:

```
10 READ X
20 PRINT "AZ X ERTEKE MOST :"; X
30 GOTO 10
40 DATA 1,34,10.5,16,234.56

READY.
```

```
AZ X ERTEKE MOST : 1
AZ X ERTEKE MOST : 34
AZ X ERTEKE MOST : 10.5
AZ X ERTEKE MOST : 16
AZ X ERTEKE MOST : 234.56
? OUT OF DATA ERROR IN 10
READY
```

A program felépítéséből következik, hogy a 10-es és 30-as sorok közötti rész többször kerül végrehajtásra. Mindannyiszor amikor a program a 10-es sorhoz ér, a READ utasítás hatására az X felveszi a DATA utasításban soron következő szám értékét. A számítógép egy ún. MUTATÓ (angolul POINTER) segítségével jegyzi meg, hogy hol tart a DATA utasításban levő számok olvasásában. Ennek a MUTATÓ-nak mindig az aktuális számra kell mutatnia, ezért nyilvánvaló, hogy értékének minden egyes programhurokban meg kell változnia és a soron következő számra kell rámutatnia.

```
         mutató
           ↓
40 DATA 1, 34, 10.5, 16, 234.56
```

Amikor a program az utolsó szám elolvasása után ismét a READ utasításhoz ér, a sikertelen olvasási kísérlet miatt, az OUT OF DATA ERROR (vége az adatoknak) szövegű hibaüzenet jelenik meg a képernyőn.

A DATA utasításnál a következő formát kell betartani:

```
40 DATA 1, 34, 10.5, 16, 234.6
          ↑                    ↖
```

Az egyes adatokat vessző választja el egymástól

Az utolsó szám után nincs vessző

A DATA parancs tartalmazhat egész, decimális vagy lebegőpontos alakban megadott számokat is, ezzel szemben sem változókat, sem aritmetikai kifejezéseket nem tartalmazhat.

Így a következő DATA sor helytelen:

```
40 DATA A,23/56,2*5
```

A DATA utasítás karakterfűzért is tartalmazhat. Ekkor azonban – ahogy azt a következő példánkban láthatjuk – a hozzá tartozó READ utasítást egy karakterfűzér változóval kell ellátnunk.

```
10 FOR X=1 TO 3
15 READ A$
20 PRINT "A$ ERTEKE MOST : "; A$
30 NEXT
40 DATA LEGYEL, MINDIG, VIDAM

READY.
```

Az utolsó példával ellentétben, itt a READ utasítás egy FOR-NEXT ciklusba van beágyazva. A ciklusmag annyiszor kerül végrehajtásra, amennyi a DATA utasításban levő elemek száma. Ha ez a szám megváltozik, értelemszerűen meg kell változnia a ciklusváltozó értékének is. Ezért sok esetben kényelmesebb, ha az adatok végét valamilyen jelzővel (szokás flagnek is nevezni) megjelöljük. Ennek a jelzőnek egy olyan elemnek (számnak vagy stringnek) kell lennie, amely adataink között még véletlenül sem fordulhat elő (pl. akármelyik negatív szám egy személyek életkorát felsoroló listán). A jelzővel összekapcsolhatunk egy feltételvizsgálatot is, amellyel egy másik programnévre való elágazást tudunk vezérelni.

Amennyiben a DATA utasításban szereplő értékeket újra fel akarjuk használni, úgy el kell érnünk, hogy a mutató ismét az első elemre mutasson. Ezt a RESTORE utasítás használatával tehetjük meg. Egészítsük ki előző programunkat a következő sor beszúrásával:

```
50 GOTO 10
```

Az így kiegészített programtól ismét az OUT OF DATA ERROR (vége az adatoknak) hibaüzenetet kapjuk. Ez nem meglepő, mivel az 50-es sor végrehajtásakor a mutató az utolsó elemre mutat, így a 10-es sor READ utasítása valóban nem találhat újabb adatot. Ezért szúrjuk be még a következő sort is:

```
45 RESTORE
```

A program így már helyes, és tetszés szerinti ideig futtatható, mivel minden egyes ciklus után a mutató a 45-ös sor eredményeképpen ismét a DATA utasítás első elemére mutat.

Középérték

A most bemutatásra kerülő program a READ és DATA utasítások egy gyakorlati alkalmazását szemlélteti. A DATA utasításból számokat olvasunk ki, majd kiszámítjuk átlagértéküket.

```
5 SZ=0 : CV=0
10 READ X
20 IF X=-1 THEN 50 : REM JELZOTESZTELES
25 CV=CV+1
30 SZ=SZ+X : REM VEGOSSZEGKEPZES
40 GOTO 10
50 PRINT CV; "ERTEKET OLVASTUNK BE"
60 PRINT "VEGOSSZEG ="; SZ
70 PRINT "KOZEPERTEK ="; SZ/CV
80 DATA 75, 80, 62, 91, 87, 93, 78, -1

READY.
```

```
RUN
7 ERTEKET OLVASTUNK BE
VEGOSSZEG = 566
KOZEPERTEK = 80.8571429
```

Az 5. sorban mind a ciklusváltozó CV, mind a számok összegét tartalmazó SZ változó értékét 0-ra állítjuk. A 10-es sorban beolvassuk az adatot. A 20-as sorban X értékét vizsgáljuk meg, vajon elértük-e a sor végét jelző flaget; példánkban ez a –1-es. Ha még nem értük el, a CV értékét megnöveljük 1-gyel (25-ös sor), X-et pedig SZ-hez adjuk (30-as sor).

Amint azonban a beolvasott X értéke a sor végét jelző –1 lesz, a program az 50-es sorra ugrik, ahol azt írja ki, hogy a középértéket hány értékből számítja ki. A 60-as sorban a számok összegét, a 70-es sorban pedig a középértéket számítjuk és nyomtatjuk ki a képernyőre. Ha tehát a DATA utasítás végét egy jelzővel zárjuk le, akkor az elemek számát tetszőlegesen növelhetjük anélkül, hogy az elemek tényleges számával törődnünk kellene.

A DATA és READ utasítások egy másik alkalmazásában a DATA-ban szereplő elemeket különböző változókhoz rendeljük hozzá. Ilyenkor – ahogy azt a következő programban láthatjuk – a DATA utasításokban a karakterfüzérek és számok tetszőlegesen keveredhetnek.

```
10 READ N$, A, B, C
20 PRINT N$; " SOROZATA :"; A; " ";         B; " "; C
30 PRINT "ENNEK ATLAGA :"; (A+B+C)/3
40 PRINT : GOTO 10
50 DATA MIKI,  190, 185, 165
60 DATA DENES, 225, 245, 190
```

```
70 DATA JANI,  155, 185, 205
80 DATA PALI,  160, 179, 187

READY.

RUN
MIKI SOROZATA : 190 185 165
ENNEK ATLAGA : 180
DENES SOROZATA : 225 245 190
ENNEK ATLAGA
```

Természetesen ilyenkor figyelnünk kell arra, hogy karakterfüzér változónak karakterfüzér, numerikus változónak pedig numerikus érték feleljen meg a DATA utasításban. Programunkban a READ utasítás először egy karakterfüzért, majd két numerikus értéket, azaz két számot vár; a DATA utasítások struktúrája ennek megfelelő.

Indexelt változók

Eddig csak közönséges változókat használtunk, amelyek alakja a következő: A, XY, K1, C$ stb. A változóneveket vagy egy betűből álló, vagy két betűből, ill. egy betűből és egy számból álló kombináció alkotja. Bár az így létrehozható változónevek mennyisége a legtöbb esetben elegendőnek bizonyul, sok esetben azonban az ilyen változók kezelése kifejezetten körülményes.

Bevezetjük ezért az ún. indexelt változó fogalmát:

```
A(1)
↑ ↑
│ └── index
└──── tömbnév
```

Az indexelt változó egy betűből és az őt követő, zárójelekbe tett számból, az ún. indexből áll. Az ilyen változót így olvasssuk ki: a egy(es). Vegyük észre, hogy A(1) és A1 különbözőek; A1 nem indexelt változó!

A közönséges változókhoz hasonlóan az indexelt változók is kijelölnek egy tárcellát a számítógépben. Az indexelt változók tárbeli képét megszámozott rekeszekként képzelhetjük el.

A(0)	0
A(1)	0
A(2)	0
A(3)	0
A(4)	0

Ha a definíció:

A(0)=25:(3)=55:A(4) = −45,3

akkor A tárbeli képe

A(0)	25
A(1)	0
A(2)	0
A(3)	55
A(4)	−45.3

Megjegyzés: Amennyiben nem adunk értéket a numerikus indexelt változónak, alapértéke 0 lesz. Létezik indexelt karakterfüzér változó is (pl. B$(2)). Az ilyen változó alapértéke – ha más értéket nem adunk neki –az "üres" lesz.

Az indexelt változók fenti halmazát TÖMB-nek nevezzük. Eddig csak az egydimenziós tömbökkel ismerkedtünk meg (az egy indexszel rendelkezőkkel), de a későbbiek folyamán rátérünk majd a többdimenziós tömbök tárgyalására is.

Az indexek összetett kifejezések is lehetnek, tartalmazhatnak pl. változót vagy valamilyen artitmetikai műveletet is. Megengedettek tehát az indexelt változók következő kifejezései:

```
A(X)  A(X+1)  A(2+1)  A(5*3)
```

A zárójelben levő kifejezésekre a 2. fejezetben ismeretett műveleti szabályok érvényesek.

Valószínűleg az alapfogalmak mostanra már tisztázódtak az Olvasóban, de felmerülhet a kérdés: mit is kezdhetünk ezekkel az indexelt változókkal?

Egy alkalmazás kézenfekvően adódik. A számokat, amelyeket eddig a READ vagy az INPUT utasítással olvastattunk be, helyezzük el egy listába. A következő programban, amelyben az eddigiektől kissé eltérő módon számíthatjuk ki a számok középértékét, indexelt változókat használunk:

```
5 PRINTCHR$(147)
10 INPUT "HANY SZAM LESZ "; X
20 FOR A=1 TO X
30 PRINT A; ". SZAM "; : INPUT B(A)
40 NEXT
50 SZ=0
60 FOR A=1 TO X
70 SZ=SZ+B(A)
80 NEXT
90 PRINT : PRINT "KOZEPERTEK = "; SZ/X

READY.
```

```
RUN
HANY SZAM LESZ : ? 5
1 SZAM ? 125
2 SZAM ? 167
3 SZAM ? 189
4 SZAM ? 167
5 SZAM ? 153
KOZEPERTEK = 161.2
```

Ezt a programozási problémát nyilván elegánsabban is megoldhattuk volna, célunk azonban itt csak az volt, hogy bemutassunk egy olyan példát, amely indexelt változókat használ. A 10-es sorban a program megkérdezi, hogy hány darab számot kell bevinnie. Az X-hez rendelt érték adja meg, hogy a ciklusnak hányszor kell futnia. Minden egyes futásnál az A index értéke eggyel nő, így a B(A)-ból folyamatosan a B(1), B(2), B(3)... stb. indexelt változók keletkeznek, amelyek aztán sorban felveszik a feldolgozandó számok értkeit. Ennek aztán az a szerencsés következménye, hogy ezekhez a számokhoz nagyon könnyen hozzáférhetünk pl. azért, hogy kinyomtassuk vagy valamilyen módon feldolgozzuk. Az 50–80-as sorokban pl. a középérték kiszámolását láthatjuk.

A számokat a következő egyszerű módon mi magunk is kiírhatjuk a képernyőre. Írjuk be a következő sort:

`FOR A = 1 TO 5 : ? B(A),: NEXT` (RETURN – kocsi vissza)

A RETURN billentyű lenyomása után a képernyőre kiírva, megkapjuk az indexelt változó tartalmát, azaz a B(1), B(2)...B(5) tömbelemek értékeit.

Dimenzionálás

Ha az előbbi példában több mint 10 értéket akarunk megadni – csupán a ciklusváltozó módosításával –, akkor a következő hibaüzenetet kapjuk: BAD SUBSCRIPT ERROR (Hibás index).

A kevesebb mint 11 elemű tömböket (amely a 0–9 indexértéknek felel meg egydimenziós tömböknél) a közönséges változókhoz hasonlóan kezelhetjük; a 11 vagy annál több elemet tartalmazó tömböket azonban minden esetben dimenzionálni kell.

Ebből a célból illesszük a következő sort az előbbi programhoz:

```
5 DIM B(100)
```

Ezáltal közöljük a számítógéppel, hogy maximálisan 100 értéket akarunk megadni. Tudunk azonban változón keresztül is dimenzionálni. Először is töröljük ki az 5-ös sort és írjuk be helyette a következőt:

```
15 DIM(X)
```

Így a tömböt pontosan akkora méretűre dimenzionáljuk, amennyi a feldolgozandó elemek száma.

Gondoljuk meg azonban: az egyszer már végrehajtott dimenzionálást a program további futása során már nem lehet érvényteleníteni, tehát a tömbök dimenzionálásakor körültekintően kell eljárnunk.

Egyszerre több indexelt változó együttes dimenzionálását a következő módon lehet összekapcsolni:

```
10 DIM C(29), D(50), E(40)
```

Kockajáték

Az indexelt változók alkalmazásával komplikált programokat is tömören és egyszerűen tudunk leírni.

A következő programban egy indexelt változó használatával fogjuk regisztrálni azt, hogy egy bizonyos szám hányszor fordul elő a kockajáték alatt.

Az F tömb F(1) ... F(6) elemei sorjában az 1-es, 2-es, ... 6-os dobások előfordulásának gyakoriságát regisztrálják; így pl., ha 2-es dobás következik be, az F(2) tömbelem értéke eggyel megnő.

```
1 REM KOCKAJATEK SZIMULACIO
2 PRINT CHR$(147)
10 INPUT "HANYAT AKAR DOBNI    "; X
20 FOR L=1 TO X
30 R=INT(6*RND(1))+1
40 F(R)=F(R)+1
50 NEXT L
60 PRINT "DOBAS","DOBASOK SZAMA"
70 FOR C=1 TO 6 :PRINTC,F(C) : NEXT

READY.
```

A program a 10-es sorban megkérdezi, hogy hány darab kockadobást kell végrehajtani; a kívánt számot az X változóban tárolja. A 20–50-es sorokban található ciklusban – amely X-szer, fog futni – a dobás szimulálását és az eredmény regisztrálását találjuk. A program a 60–70-es sorokban az eredményeket írja ki táblázatos formában. A program futása után a képernyő valahogy így fog kinézni:

```
Hányat akar dobni
```

Dobás	Dobások száma
1	157
2	153
3	173
4	188
5	152
6	177

A következőkben egy olyan programot adunk közre, amely ugyanazt nyújtja, mint az előző program, de anélkül, hogy indexelt változót használna. Nem szükséges, hogy beírjuk a programot, de figyeljük meg, hogy végrehajtása milyen hosszadalmas az előző programhoz képest!

```
10 INPUT "HANYAT AKAR DOBNI ";X
20 FOR I=1 TO X
30 R=INT(6*RND(1))+1
40 IF R=1 THEN F1=F1+1 : NEXT
41 IF R=2 THEN F2=F2+1 : NEXT
42 IF R=3 THEN F3=F3+1 : NEXT
43 IF R=4 THEN F4=F4+1 : NEXT
44 IF R=5 THEN F5=F5+1 : NEXT
45 IF R=6 THEN F6=F6+1 : NEXT
60 PRINT "DOBAS","DOBASOK SZAMA"
70 PRINT 1, F1
71 PRINT 2, F2
72 PRINT 3, F3
73 PRINT 4, F4
74 PRINT 5, F5
75 PRINT 6, F6

READY.
```

A program hosszúsága a 8-astól a 16-os sorig megduplázódott. Nagyobb programoknál, különösen akkor, ha a változók száma nagy, az indexelt változókkal még több sort és ezzel együtt még több tárcellát tudunk megtakarítani.

Kétdimenziós tömbök

A kétdimenziós tömb elemeinek formája a következő:

```
A (4,6)
↑  ↑ ↑
|  indexek
tömbnév
```

Egy ilyen tömböt szemléletesen kétdimenziós rácsként lehet elképzelni.

	0	1	2	3	4	5	6
0							
1							
2							
3							
4							

Az indexek itt azt adják meg, hogy a kérdéses elem a táblázat melyik sorában és melyik oszlopában található meg.

Az ábra alapján könnyen magunk elé képzelhetjük, hogy a 255-ös elem táblázatunk 3. sorában és 4. oszlopában helyezkedik el.

A(3,4) = 255

↑ SOR   ↑ OSZLOP

	0	1	2	3	4	5	6
0							
1							
2							
3					255		
4							

Akárcsak az egydimenziós tömböket, úgy a többdimenziósokat is dimenzionálni kell. Pl. egy 21-szer 21-es elemű tömböt a DIM A(20,20)-szal kell dimenzionálni. Mi tehát egy kétdimenziós tömb tipikus felhasználása?

Tegyük fel, hogy egy kérdőívet kell megszerkeszteni, amely 4 olyan kérdést tartamaz, amelyekre egyenként 3 különböző válasz lehetséges! A kérdőívünket a következőképpen építhetjük fel:

1. KÉRDÉS: Elégedett Ön a főnökével?
   1=IGEN 2=NEM 3=KÖZÖMBÖS ... stb.

A kérdőívek értékelését az alábbi táblázat alapján képzelhetjük el. A táblázatot egy kétdimenziós tömb tárbeli képének foghatjuk fel.

	VÁLASZOK		
	IGEN	NEM	KÖZÖMBÖS
1 KÉRDÉS			
2 KÉRDÉS			
3 KÉRDÉS			
4 KÉRDÉS			

A hozzá tartozó kiértékelő programot is megadjuk. Ebben a programban többféle olyan programozási technikát használtunk fel, amelyeket eddig megbeszéltünk. Ezért még akkor is ajánlatos megérteni a program felépítését, ha nem óhajtjuk a későbbiekben használni. A program magját egy 4*3 elemes tömb alkotja. Az elemekben tartjuk számon a válaszokat. Az áttekinthetőség kedvéért az A(0,0)-tól az A(0,4)-ig terjedő elemeket nem használjuk.

A program futásakor: Ha az egyes kérdésre IGEN a válasz, akkor az A(1,1) értéke nő meg 1-gyel: a táblázatban az első sor, ill. az első oszlop lesz kiválasztva, mivel a sorszámot a kérdés száma, az oszlopszámot pedig a válasz milyensége (IGEN, NEM, KÖZÖMBÖS) határozza meg. További példaként: a 3-as sorszámú kérdésre adott NEM válasz, az A(3,2)-es elemek értékét növeli meg eggyel.

```
20 PRINT "□" : REM SHIFT+CLR BILLENTYUK
30 FOR K=1 TO 4
40 PRINT "A KERDES SZAMA :" ; K
50 PRINT "1-IGEN 2-NEM 3-KOZOMBOS"
60 PRINT "A VALASZ : ";
61 GET C : IF C<1 OR C>3 THEN 61
65 PRINT C : PRINT
70 A(K,C)=A(K,C)+1
80 NEXT K
85 PRINT
90 PRINT "AKAR MEG VALASZOKAT MAGADNI?"
91 PRINT "VALASZ(I/N)"
100 GET A$ : IF A$="" THEN 100
110 IF A$="I" THEN 20
120 IF A$<>"N" THEN 100
130 PRINT "□AZ OSSZES VALASZ :" : PRINT
140 PRINTSPC(18); "VALASZ"
141 PRINT "KERDES","IGEN","NEM","KOZOMBOS"
142 PRINT "------   ---------------------------------"
150 FOR K=1 TO 4
160 PRINT K, A(K,1), A(K,2), A(K,3)
170 NEXT K

READY.
```

```
RUN
A KERDES SZAMA : 1
1-IGEN 2-NEM 3-KOZOMBOS

A KERDES SZAMA : 2
1-IGEN 2-NEM 3-KOZOMBOS

A KERDES SZAMA : 3
1-IGEN 2-NEM 3-KOZOMBOS
AZ OSSZES VALASZ
```

	VALASZ		
kerdes	igen	nem	kozombos
1	7	1	0
2	6	2	0
3	8	0	0
4	2	5	1

FÜGGELÉK

Bevezetés

Bár mostanra már szoros kapcsolatba kerültünk COMMODORE 64-esünkkel, ha akarjuk folytathatjuk az ismerkedést, hiszen a könyv még sok érdekességet rejteget. A 70-es években jelent meg a COMMODORE első eredeti személyi számítógépe, a PERSONAL ELECTRONIC TRANSACTOR. A cég azóta a világ legtöbb országában a vezető számítógépgyártók sorába lépett. A COMMODORE cég saját maga fejleszti és gyártja a számítógépeihez használt chipeket. A következő Függelékek ábrákat, táblázatokat és egyéb kiegészítő informciókat tartalmaznak. Mindezek segíthetnek abban, hogy COMMODORE 64-esünket gyorsabban és hatékonyabban programozhassuk.

A COMMODORE 64-es tartozékai és a szoftver

A COMMODORE 64-es VC–20-as tárolóegységet és további tartozékokat használ: kazettásegységet, lemezmeghatjó egységet és nyomtatót.

Kazettás adattrögzítő egység: Ezzel az olcsó szalagos egységgel programokat és adatokat rögzíthetünk a közönséges, kereskedelemben is kapható magnókazettákra. Kész programokat is vásárolhatunk ilyen kazettán.

Lemez – A VC 1541-es típusú lemezmeghajtó egység 5 1/4 inch-es standard lemezt használ programok és adatok tárolására. A lemezesegység gyorsabb adathozzáférést tesz lehetővé, és minden lemezen 170000 karaktert képes tárolni. Az egység intelligens, saját mikroprocesszorral és saját tárral rendelkezik, vagyis nem igényel külön munkaterületet a számítógép tárában.

Nyomtató – A VC–1525-ös típusú nyomtató programok, adatok és rajzok kinyomtatott másolatát hozza létre. Ez a mátrixnyomtató másodpercenként 30 karaktert dolgoz fel, és a nyomtatáshoz közönséges papírt vagy más, hasonlóan olcsó anyagot használ.

Csatlakoztatható interface kártyák – A COMMODORE 64-eshez nagyszámú bedugható interface kártya áll rendelkezésre, amelyekkel olyan különböző alapeszközökhöz lehet kapcsolódni, mint a nyomtatók, a modemek, vezérlők, mérőeszközök és egyéb műszerek.

Egy speciális IEEE–488-as csatlakoztatható interface kártyával az összes CBM perifériát használhatjuk, még a lemezeket és nyomtatókat is. Ezenfelül egy Z80-as kártya lehetővé teszi, hogy gépünkkel a CP/M operációs rendszer alatt is dolgozhassunk. Ily módon hozzáférhetővé válik a mikroprocesszor alkalmazások széles skálája.

A szoftver

A COMMODORE 64-eshez különböző kategóriájú programokat kínálnak. A programok széles skálájából: a háztartás, a személyes adatnyilvántartás, a szórakozás, az oktatás stb. területéről válogathatunk.

Segítség a napi munkában

- Egy Elektronikus Könyvelőív programcsomag lehetővé teszi, hogy költségvetési terveket készíthessünk és "mi lenne ha?" típusú kérdéseket elemezhessünk. A programcsomaghoz tartozó grafikus programmal az ilyen adatlapokból látványos grafikonokat készíthetünk.

- Pénzügyi tervezéseket lehetővé tevő programcsomaggal könnyedén megvalósíthatunk olyan pénzügyi jellegű tervezéseket, mint pl. beruházások amortizációja stb.
- Munkaidőnk, határidőink és egyéb tevékenységeink legcélszerűbb összehangolásában nagy segítséget nyújthat számunkra egy sor "időbeosztó" program.
- Könnyen használható adatbáziskezelő programok fontos információinkat – telefonlistákat, leltári jegyzékeket, szállítási kartotékokat – a számunkra legmegfelelőbb formában, naprakészen tartják nyilván.
- A professzionális szövegszerkesztő programok COMMODORE 64-esünket teljes értékű titkárnői munkahellyé változtathatják. Az olyan egyébként fáradságos tevékenységeket, mint a feljegyzések, levelek vagy egyéb szöveganyagok szerkesztése és módosítása, a programmal könnyedén elvégezhetünk.

Szórakozás

A legérdekesebb játékokhoz, csatlakoztatható modulokon juthatunk hozzá. Ezek teljes mértékben kiaknázzák a COMMODORE 64-es nagyfelbontású grafikájának lehetőségeit, valamint a gép zaj- és zenélőképességét is.

Oktatás

A COMMODORE 64-es olyan tanár, amely sohasem fárad el és mindig ránk figyel. Számítógépünkkel a sokféle CBM oktatóprogram mellett, a PILOT, a LOGO és más haladó szintű programcsomag használata is lehetővé válik.

B FÜGGELÉK

Műveletek a kazettásegységgel

Azt az elrendezést, amelyben a COMMODORE 64-es a programok másolása mellett a különböző értékeket, változókat és az adatok más fajtáit a mágnesszalagra viszi, ÁLLOMÁNY-nak nevezzük. Az állományok annál több információt tudnak tárolni, minél több adat helyezhető el egyszerre a számítógép tárában.

Az adatállományokkal kapcsolatos utasítások a következőek: OPEN, CLOSE, PRINT#, INPUT# és GET#.

Az ST (sátusz) rendszerváltozót pl. arra használhatjuk, hogy a mágnesszalagra vett jeleket ellenőrizzük.

Az adatok mágnesszalagra vitelénél ugyanazt a koncepciót használjuk, mint a képernyőre vitelnél, csak az adatfolyam most a magnetofonra irányul. Ez felismerhető a PRINT utasítás erre a célra használt változatánál, nevezetesen, PRINT#-nál.

Az elmondottakat a következő program szemlélteti:

```
10 PRINT "SZALAGRA IRAS"
20 OPEN 1,1,1, "ADATALLOMANY"
30 PRINT "IRJA BE AZ ADATOKAT VAGY A STOP SZOT !"
50 PRINT
60 INPUT "ADAT"; A$
70 PRINT# 1, A$
80 IF A$<>"STOP" THEN 50
90 PRINT
100 PRINT "ALLOMANY LEZARVA"
110 CLOSE 1

READY.
```

Elsőként egy állományt kell megnyitnunk (esetünkben a 20-as sorban, ADATÁLLOMÁNY név alatt).

A program a 60-as sorban a tárolni kívánt adatokat kéri, a 70-es sorban pedig kiviszi őket a mágnesszalagra (a beírt karaktereket az A$ változó tárolja). Ezután ismét adatokat kér.

Ha már nem akarunk több adatot a mágnesszalagra vinni, írjuk be a STOP szót! Ennek hatására a program lezárja az állományt. Azért, hogy az adatokhoz ismét hozzájussunk, tekercseljük vissza a szalagot és futtassuk le a következő programot:

```
10 PRINT "SZALAGOLVASO PROGRAM"
20 OPEN 1,1,0, "ADATALLOMANY"
30 PRINT "ALLOMANY NYITAS"
40 PRINT
50 INPUT# 1, A$
```

```
60 PRINT A$
70 IF A$="STOP" THEN CLOSE 1 : END
80 GOTO 40

READY.
```

Az ADATÁLLOMÁNY-t ismét meg kell nyitni. Az 50-es sorban az A$ változót beolvassuk (INPUT#), a 60-as sorban pedig kiírjuk a képernyőre (PRINT). Az eljárás egészen a STOP szó beolvasásig ismétlődik, amikor is a program befejeződik.

A GET utasítás egyfajta változatával, nevezetesen a GET#-el, az adatok ugyancsak leolvashatók a szalagról. Helyettesítsük az előbbi program 50–80-as sorait a következő programrésszel:

```
50 GET# 1, A$
60 IF A$="" THEN CLOSE 1 : END
70 PRINT A$, ASC(A$)
80 GOTO 50

READY.
```

C FÜGGELÉK

BASIC

Kézikönyvünk eddig csak bevezetőt nyújtott a BASIC nyelvbe, arra elegendőt, hogy megérthessük a számítógép programozását és a megadott BASIC nyelvű programokat. Ebben a fejezetben megtaláljuk a COMMODORE 64-es BASIC nyelv szabályainak (szintaxisának) teljes listáját, rövid magyarázatokkal kiegészítve. A parancsokat egyszerű módon próbáljuk is ki! Egy program beírásával ugyanis semmilyen tartós kárt nem okozhatunk számítógépünkben, tanulni viszont programozás közben lehet leghatékonyabban. A BASIC különböző típusú műveleteinek megfelelően a C Függelék több részre oszlik:

1. **Változók és operátorok:** a különböző változótípusok, az érvényes változónevek, az artimetikai és logikai operátorok leírásával foglalkozik.
2. **Parancsok:** a programok szerkesztéséhez, hívásához, tárolásához és törléséhez szükséges parancsok leírását tartalmazza.
3. **Utasítások:** a sorszámozott programsorokban előforduló BASIC utasításokat írja le.
4. **Függvények:** a string-, a numerikus- és a nyomtatási funkciók leírását foglalja magába.

A változók

A COMMODORE 64-es a változók három fajtáját használja: a lebegőpontos (real=valós), az egész (integer) és az alfanumerikus karakterfüzér (string) típusúakat.

A változónevek állhatnak egy egyedül álló betűből, egy betűből és egy rögtön utána következő számból, valamint két egymás mellett levő betűből.

Az egész típusú változók neveit a százalékjel (%), még a karakterfüzér típusú változók neveit a mögéjük tett dollárjel ($) különbözteti meg.

Példák:

Valós típusú változónevek:	A,A5,BZ
Egész típusú változónevek:	A%,A5%,BZ%
Karakterfüzér típusú változónevek:	A$,A5$,BZ$

A tömbök olyan változókból állnak, ahol az egyes változók nevei ugyanazok, de egymástól, a tömb egyes elemeiként, egy speciális szám hozzárendelésével különülnek el. A tömböket a DIM statement-tel kell definiálni (a statement a programozástechnikában gyakran használt angol szó, a parancs, állítás szavak

megfelelőjeként). A tömbök elemei mindhárom, azaz valós, egész és karakterfüzér típusúak is lehetnek. A változóneveket egy zárójelbe tett adat követi, amely azt adja meg, hogy a lista hány elemet tartalmaz, és hogyan kell ezeket elrendezni (vagyis hány dimenziós a tömb).

A(7), BZ%(11), A$(50), PT(20,20)

Megjegyzés: Van három olyan változónév, amelyet a COMMODORE 64-es maga használ: ST, TI és a TI$, ezeket nem definálhatjuk. Az ST egy státuszváltozó, amelynek aktuális tartalma az előző be-, ill. kiviteli műveletnek megfelelően változik. Pl. az ST akkor változtatja meg értékét, ha a mágnesszalag vagy a mágneslemez írásakor vagy olvasásakor valamilyen probléma merül fel.

A TI értékét a belső óra 1/60 másodpercenként növeli. A számítógép bekapcsolásakor értéke 0, és csak akkor nullázható ismét, ha a TI$ értékét megváltoztatjuk.

A TI$ egy 6 számjegyű karakerfüzér, amelyet maga a rendszer állít be. Az első két számjegy az órákat, a második kettő a perceket, míg az utolsó kettő a másodperceket jelöli. A hat számjegynek tetszőleges kezdeti értéket adhatunk, amelyeket azután a rendszer folyamatosan továbbléptet.

TI$="101530" beállítás 10 óra 15 perc 30 másodpercet jelent. Az óra a számítógép kikapcsolásakor törlődik és bekapcsolásakor 0-ról kezd számoni.

Az operátorok

Az aritmetikai operátorokat a következő műveleti jelek képezik:

+ összeadás
– kivonás
* szorzás
/ osztás
↑ hatványozás

Ha egy sorban több műveleti jelet használunk, akkor a műveletek sorrendjére a matematikában is ismert szabályok érvényesek. Azaz, elsőként a hatványozás, majd a szorzás és az osztás, legvégül pedig az összeadás és a kivonás műveletét kell elvégezni.

A műveletek kiszámítási sorrendjét a kerek zárójelekkel befolyásolhatjuk, ugyancsak a matematikából megismert módon, a zárójelekben levő műveleteket végezzük el elsőként.

Összehasonlítható műveleteket a következő operátorokkal fogalmazhatunk meg:
= egyenlő
< kisebb mint
> nagyobb mint
<= kisebb egyenlő mint

>= nagyobb egyenlő mint
<> nem egyenlő

Végül, a logikai operátorok:

AND	ÉS (két feltételnek egyszerre kell teljesülnie)
OR	VAGY (a két feltételből legalább az egyiknek teljesülnie kell)
NOT	NEM (a feltételnek nem kell teljesülnie)

Ezeket általában többszörös feltételek megfogalmazására használjuk az IF... THEN utasítások feltétel rendszerében. Pl.: IF A = B AND C = D THEN 100 (legalább az egyik feltételnek teljesülnie kell).

Parancsok

CONT (Continue)

Ezt a parancsot arra használjuk, hogy a STOP billentyű, a STOP utasítás vagy az END utasítás hatására megállt programot továbbindítsuk. A program pontosan azon a helyen folytatódik, ahol megszakadt. Ha azonban egy előtte álló programsort megváltoztatunk vagy beszúrunk a CONT utasítás nem működik. Ugyanígy akkor sem működik a CONT, ha valamely hiba folytán a programmegszakítást töröltük vagy valamilyen más hibát okoztunk előzőleg. Ilyen esetekben a következő hibaüzenetet kapjuk: CAN'T CONTIUNE ERROR (Nem folytatható).

LIST

A LIST használatával programsorokat tudunk megjelentetni. Kiírhatjuk akár az egész programot vagy csak annak egy kiválasztott részét.

LIST	az egész programot kiírja
LIST 10–	a 10-es sortól a végéig terjedő részt jeleníti meg
LIST 10	csak a 10-es sor jelenik meg
LIST –10	a 10-es sorig terjedő részt írja ki
LIST 10–20	a 10–20-as sorok jelennek meg

LOAD

Ezt a parancsot akkor használjuk, ha a mágnesszalagról vagy a mágneslemezről programot akarunk betölteni a számítógépbe. Amennyiben csak a LOAD parancsot írjuk be, úgy a RETURN billentyű lenyomása után a kazettáról a soron következő programot tölti be. A parancsot azonban egy idézőjelek között levő programnévvel kiegészíthetjük. A nevet még követheti egy vessző, továbbá egy szám vagy egy

numerikus változó is, amelyeket a számítógép a betöltés forrásának azonosítójaként értelmez. Innen kell a program betöltését elvégeznie.

Ha nincs eszközszám megadva, a COMMODORE 64-es ezt 1-nek veszi, azaz a kazettásegységnek. A másik szokásos eszközszám a 8-as, amely a lemezmeghajtó egységet azonosítja.

`LOAD`	a következő programot olvassa be a kazettáról
`LOAD"HALLO"`	a kazettán megkeresi a HALLO nevű programot majd ha megtalálta, betölti
`LOAD A$`	azt a programot keresi meg és tölti be, amelynek neve az A$ változóban van
`LOAD "HALLO",8`	a HALLO nevű programot betölti a lemezről
`LOAD"*",8`	a lemezen található első programot tölti be

NEW

A NEW parancs törli a tárban levő programot, és az utána elhelyezkedő öszes változót is.

Ha a SAVE utasítással nem tároltuk a programot, elveszik. Óvatosan használjuk ezt az utasítást!

A NEW parancsot programban is használhatjuk. Ha azt szeretnénk elérni, hogy a számítógép tára üres legyen a program futása után, helyezzünk el egy NEW utasítást a program végén!

RUN

A számítógépbe betöltött programok végrehajtását a RUN paranccsal kezdeményezhetjük. A szám nélkül megadott RUN kulcsszó a program végrehajtását a legalacsonyabb sorszámú sorban kezdi. A RUN után megadott szám a legelsőnek végrehajtandó programsor sorszámot azonosítja.

`RUN`	a program futása a legalacsonyabb sorszámú sorban kezdődik
`RUN 100`	a program a 100-as sor végrehajtásával kezdődik
`RUN X`	UNDEFINED STATEMENT ERROR (Definiálatlan utasítás). A megadott sorszámnak léteznie kell és explicit módon kell megadni. A parancsban nem használhatunk változót.

SAVE

Lemezre vagy szalagra tárolja az éppen a tárban levő programot. Ha csak a SAVE kulcsszót írjuk be, majd RETURN-t nyomunk, akkor a program a szalagra kerül. A számítógép nem tudja megállapitani, hogy a szalagnak ezen a részén van-e már program, ezért legyünk óvatosak, nehogy felülírjuk korábban tárolt értékes programjainkat!

Ha a SAVE után egy idézőjelbe tett nevet, ill. egy karakterfüzér változót is megadunk, akkor programunk ezzel a névvel íródik a szalagra, így a későbbiekben könnyen visszakereshetjük.

A név után vesszővel elválasztva még két szám, a készülékszám és a másodlagos cím (szalagról 0 vagy 1) állhat. Ha a másodlagos cím 1, akkor a gép a tárolás után egy szalagvég jelet ír a szalagra. Ha egy LOAD parancs végrehajtása közben a gép egy ilyen jelbe ütközik, mielőtt a kívánt programot megtalálta volna, akkor a FILE NOT FOUND (Nincs meg az állomány) üzenetet írja a képernyőre.

`SAVE`	névtelenül tárolja a programot a szalagra
`SAVE"HALLO"`	a program neve HALLO lesz a szalagon
`SAVE A$`	a program neve az A$-ben levő karakterfüzér lesz
`SAVE"HALLO",8`	HALLO névvel tárolja a programot a lemezre
`SAVE"HALLO",1,1`	HALLO névvel tárolja a programot a szalagra, majd elhelyez egy szalagvég jelet

VERIFY

A parancs hatására a gép ellenőrzi, hogy szalagon vagy lemezen levő program megegyezik-e a tárban levővel, így lehetővé válik a tárolás során fellépő hibák kiszűrése.

Ha a VERIFY kulcsszó után semmi mást nem írunk, akkor a gép a szalagon soron következő programot hasonlítja össze a tárban levővel. Egy konkrét program ellenőrzését a név és a készülékszám megadásával írhatjuk elő.

`VERIFY`	ellenőrzi a szalagon soron következő programot
`VERIFY"HALLO"`	ellenőrzi a szalagon a HALLO nevű programot
`VERIFY"HALLO",8`	ellenőrzi a lemezen a HALLO nevű programot

Utasítások

CLOSE

Lezárja az OPEN-nel megnyitott adatállományokat. A CLOSE kulcsszó után a lezárandó állomány azonosítóját (számát) kell írnunk.

`CLOSE 2` lezárja a #2-es adatállományt

CLR (CLeaR)

Törli a tárban levő összes változót, magát a programot azonban érintetlenül hagyja. RUN parancs kiadásakor automatikusan végrehajtásra kerül.

CMD (CoMmanD)

Az alaphelyzetben a képernyőre irányuló adatfolyamot (pl. PRINT, LIST) egy másik készülékhez rendeli hozzá (kivétel: képernyőírás POKE-kal). Ez a készülék lehet pl. a nyomtató, ill. egy a szalagon vagy a hajlékonylemezen levő adatállomány. A CMD kulcsszót az állományt azonosító számnak vagy numerikus változónak kell követnie.

`OPEN 1,4`	megnyitja a #4-es készüléket (nyomtató)
`CMD 1`	a tárolandó adatfolyam mostantól a nyomtatóra irányul
`LIST`	a programlista most nem a képernyőre, hanem a nyomtatóra kerül

DATA

A DATA kulcsszót a READ utasítással kiolvasható adatsorozat követi. Az egyes tételeket, amelyek numerikus vagy karakterfüzér típusúak lehetnek, vessző választja el egymástól. A szöveges adatokat nem kell idézőjelbe tenni, hacsak nem tartalmaznak betűközt, vesszőt vagy kettőspontot. Ha két elválasztó vessző között nem áll semmi, akkor innen numerikus értékként 0-t, karakterfüzérként pedig üres karaktert olvas be a READ utasítás.

```
10 DATA 12, 14.5,"HALLO,FIUK",3.14 HALLO
```

DEF FN (DEFine FuNction)

Lehetővé teszi, hogy az összetett számításokat egy rövid névvel azonosított függvényként definiáljuk. A terjedelmes formulák esetében ilyen módon helyet és időt takaríthatunk meg.

A függvényneveknek FN-nel kell kezdődniük, ezt követi 1 vagy 2 általunk választott karakter. A függvényeket a DEF utasítás definiálja, amelyet a függvény neve követ. A név után zárójelben egy numerikus változó áll. Ezután egy egyenlőségjel, majd az előbbi numerikus változóra épülő kiszámítandó formula következik. A függvényt ezután a neve leírásával hívhatjuk, a zárójelbe azt a számot vagy numerikus változót téve, amelyet a definiáláskor megadott formulába kívánunk helyettesíteni.

```
10 DEF FNA(X) = 12*(34.75-X/.3)
20 PRINT FNA(7)
```

Az eredmény ebben az esetben 137 lesz.

DIM (DIMension)

Ha egy tömbben 11-nél több elemet akarunk tárolni, akkor a tömbnek a DIM utasítással kell helyet foglalnunk a tárban. A felesleges tárfoglalás elkerülésére ne definiáljunk szükségtelenül nagy méretet.

Az egyes tömbök által lefoglalt változóterületet az indexek összeszorzásával kaphatjuk meg.

```
10 DIM A$(40), B7(15), CC%(4,4,4)
           ↑        ↑        ↑
       41 elem   16 elem   5*5*5=125 elem
```

Egy DIM utasítással több tömböt is definiálhatunk, ugyanaz a tömb viszont csak egyszer dimenzionálható.

END

Ha a program ráfut egy END utasításra, akkor a végrehajtás befejeződik. A program CONT-tal újraindítható.

FOR...TO...STEP

A NEXT utasítással együtt lehetővé teszi, hogy ugyanazt a programrészt többször is futtassuk. Pontos formája:

FOR (Változónév) = (Ciklusváltozó kezdőértéke) TO (Ciklusváltozó végértéke) STEP (lépésköz).

A ciklus minden futása után a ciklusváltozó értéke a STEP után megadott lépésközzel növekszik, ill. csökken, ha a lépésköz negatív szám. Ha a STEP kulcsszót elhagyjuk, akkor a lépésköz értéke automatikusan 1 lesz.

```
10 FOR C=1 TO STEP .1
20 PRINT C
30 NEXT C
```

Ha a ciklusváltozó értéke eléri a végértékként megadott határt, akkor a program kilép a ciklusból.

GET

A GET-tel egy karaktert olvashatunk be a billentyűzetről. A bevitt karakter az utasításban megadott változóba kerül. Ha semmilyen billentyűt sem ütünk le, akkor a változó értéke 0 vagy üres karakter lesz.

A GET-tel általában egy karakterfüzér típusú változóba olvasunk be adatot. Ha mégis numerikus változót használunk és nem numerikus adatot viszünk be, akkor hibaüzenetet kapunk.

A GET utasítást célszerű lehet egy ciklusba foglalni, amely ellenőrzi, hogy nyomtunk-e le valamilyen billentyűt és csak utána engedi tovább a programot:

```
10 GET A$: IF A$ = "" THEN 10
```

GET#

Ezzel az utasítással egy karaktert olvashatunk be egy korábban OPEN-nel megnyitott állományból.

```
GET# 1, A$
```

ebbe a változóba kerül a beolvasott karakter az adatállomány azonosítója

GOSUB (GOTO SUBrutine)

Ha a program ráfut egy GOSUB utasításra, akkor a GOSUB kulcsszó után megadott sorra ugrik és ott folytatja a végrehajtást. Ha ezután egy RETURN-t talál, akkor visszatér a közvetlenül a GOSUB-ot követő utasításra és innen indul tovább.

Ez a mechanizmus lehetővé teszi, hogy egy programrészt a végrehajtás különböző fázisaiban többször is futtassunk. A többszörös beírás helyett így elegendő a kívánt részt (szubrutint) egy GOSUB utasítással meghívni.

```
20 GOSUB 800
```

GOTO vagy GO TO

A program a GOTO után megadott sorra ugrik és onnan folytatja a végrehajtást. Ha a megadott sor nem létezik, akkor hibaüzenetet kapunk és a program leáll.

IF...THEN

Az IF...THEN utasítással a program egy feltétel teljesülésétől függően két lehetséges tevékenység között választ. Ha a feltétel teljesül, akkor a THEN-t követő utasítást hajtja végre, ellenkező esetben pedig a következő sorra ugrik.

A feltétel egy változó, egy aritmetikai kifejezés vagy egy logikai kifejezés lehet. Ha a változó vagy aritmetikai kifejezés aktuális értéke nem 0, akkor a feltétel teljesül, ha 0, akkor pedig nem teljesül. Logikai kifejezés esetén az igazságérték (igaz, hamis) dönt. A logikai kifejezés a következő logikai műveleteket tartalmazhatja: =, <, >, <=, >=, <>, AND, OR, NOT.

```
10 IF X>0 THEN END
```

INPUT

Az INPUT utasítás a billentyűzetről bevitt adatot egy változóhoz rendeli hozzá. A program egy (?) kérdőjelet ír a képernyőre, megáll, és mindaddig vár, amíg a választ be nem írjuk. A bevitelt a RETURN billentyűvel kell lezárnunk.

Az INPUT kulcsszó után egy változónév vagy egy vesszőkkel elválasztott változónév lista áll. A lista elé (a listától (;) pontosvesszővel elválasztva) üzenetet is írhatunk, amely a kérdőjellel együtt jelenik meg a képernyőn. Ha egy INPUT utasítással több adatot viszünk be, akkor vesszővel kell elválasztanunk.

```
10 INPUT "KEREM A NEVET";A$
20 PRINT "MOST A SZEMELYI SZAMOT KEREM" : INPUT B
```

INPUT#

Csak abban tér el az INPUT-tól, hogy az adatokat egy korábban OPEN-nel megnyitott állományból veszi.

LET

Az értékadási műveleteket jelölheti ki. Használata nem kötelező, ezért általában ritkán fordul elő a BASIC programokban.

```
10 LET A=5
20 LET D$="HELLO"
```

NEXT

Mindig egy FOR...TO-val megnyitott ciklust zár le. Amikor a program a NEXT utasításhoz ér, megvizsgálja, hogy a ciklusváltozó értéke elérte-e már a TO után megadott határt. Ha nem, akkor a ciklus a ciklusváltozó lépésközzel megnövelt értékével újra fut. Ha igen, akkor a program végrehajtása a NEXT-et követő utasítással folytatódik. A NEXT kulcsszó után egy változónév vagy egy vesszőkkel elválasztott változónév lista állhat. Mindig tartsuk be a ciklusok egymásba ágyazásának szabályait!

```
10 FOR X=1 TO 100 : NEXT
```

ON

A programok többszörös elágaztatására alkalmas. Az ON utáni artimetikai kifejezést a program értékeli, és az eredmény alapján választja ki azt a sort, amellyel a végrehajtást folytatja. Ha az eredmény 0, vagy nagyobb, mint a lista elemszáma, akkor az ON-t követő utasításra kerül a vezérlés.

```
10 INPUT X
20 ON X GOTO 10,20,30,40,50
```

OPEN

Megnyit egy adatállományt. Az OPEN kulcsszót a logikai állományszám követi, amelynek alapján a későbbi utasítások az illető állományt azonosítják. Ezután általában egy újabb szám, a készülékszám áll.

A készülékszámok:

0 : Billentyűzet
1 : Szalag
3 : Képernyő
4 : Nyomtató (vagy 5)
8 : Lemez (vagy 9–15)

A készülékszámot ugyancsak vesszővel elválasztva gyakran egy harmadik szám, a másodlagos cím követi. A szalag esetében ez olvasásra 0, írásra 1 és 2 ha szalagvég jellel akarunk írni.

A lemeznél a másodlagos cím a puffer- vagy csatornaszámra vonatkozik, a nyomtatónál pedig a különböző beállításokat vezérli.

`10 OPEN 1,0`	készülékként nyitja meg a billentyűzetet
`20 OPEN 2,1,0,"D"`	megnyitja a szalagot olvasásra, a keresett állomány D
`30 OPEN 3,4`	megnyitja a nyomtatót
`40 OPEN 4,8,15`	megnyitja a lemez parancscsatornáját

Lásd még: CLOSE, CMD, GET#, INPUT#, PRINT#, ST rendszerváltozó, B Függelék

POKE

Mindig két szám, ill. kifejezés követi. Az első egy tárcella címe, a második pedig egy 0 és 255 közötti érték, amely a POKE utasítás hatására az adott tárcellába kerül, felülírva a korábbi tartalmat.

```
10 POKE 53281,0
20 S=4096*13
30 POKE S+29,8
```

PRINT

A kezdők kedvenc utasítása, amely az adatok képernyőn való megjelenítésére alkalmas.

A PRINT után állhat:

- idézőjelbe tett karakterfüzér
- változónév
- függvények
- írásjelek

Az írásjelekkel a kiíratási formát határozhatjuk meg. A vessző hatására a kiírás a következő zónában folytatódik (a képernyőn négy zóna van), míg a pontosvessző elfojtja a betűközöket két kiírás között. Ilyenkor a következő PRINT utasítást az előző folytatásának tekinti a program.

```
10 PRINT "HELLO"
20 PRINT "HELLO",A$
30 PRINT A+B
40 PRINT J;
60 PRINT A,B,C,D
```

Lásd még: POS, SPC és TAB függvény

PRINT#

Némileg eltér a PRINT utasítástól. Egy szám követi, amely egy korábban OPEN-nel megnyitott készüléket vagy adatállományt azonosít. Utána egy vessző, majd a kiíratandó lista következik. A listában a vessző és a pontosvessző hatása ugyanaz, mint a PRINT esetében volt. Ne feledkezzünk meg arról, hogy néhány készülék a TAB-bal és az SPC-vel nem tud mit kezdeni.
Figyelem: A PRINT# nem rövidíthető ?#-tel!

```
100 PRINT#1,"ADATOK";A%,B1,C$
```

READ

A READ utasítással egy adatot olvashatunk be a DATA sorokból a megadott változóba. Ha numerikus változóba karakterfüzért próbálunk meg tévedésből beolvasni, akkor a TYPE MISMATCH ERROR (Nemegyező típus) hibaüzenetet kapjuk.

REM (REMark)

A REM után álló szöveg a programlista olvasójának szól, és általában a programra vonatkozó magyarázatokat tartalmaz. Azonkívül hogy megnöveli a program hosszát, semmilyen hatása sincs.

RESTORE

A DATA sorok beolvasásra kerülő adatára irányuló mutatót visszaállíthatja az első DATA utasítás első adatára, így lehetővé teszi a többszöri beolvasást.

RETURN

Az alprogramok (szubrutinok) lezárására alkalmas. Amikor a program egy RETURN-höz ér, akkor visszaugrik a közvetlenül a GOSUB után következő utasításra és onnan fut tovább. Ha a RETURN előtt nem hajtottunk végre GOSUB-ot, akkor a RETURN WITHOUT GOSUB (GOSUB nélküli RETURN) hibaüzenetet kapjuk.

STOP

Megállítja a program végrehajtását. Ekkor a BREAK IN xxx (Megállás az xxx-es sorban) üzenet jelenik meg a képernyőn, ahol xxx a STOP-ot tartalmazó sor száma. A program CONT-tal továbbfuttatható. Rendszerint a programok belövésénél, ill. a hibakeresésnél használjuk.

SYS (SYStem)

A SYS kulcsszó után egy 0–65535 értékű szám vagy numerikus kifejezés áll. A program végrehajtja az adott címen kezdődő gépi kódú rutint. Hasonlít a USR függvényhez, de nem tesz lehetővé paraméter-átadást.

WAIT

Mindaddig feltartja a program végrehajtását, amíg a megadott tárcímre egy meghatározott érték nem kerül. A kulcsszót egy tárcím (X) és egy vagy két változó követi:

WAIT X,Y,Z

A gép először EXOR (kizáró VAGY) műveletet végez az X tárcella tartalma és Z (ha van) között, majd AND (2S) műveletet az eredmény és Y között. Ha

a végeredmény 0 akkor a vizsgálat elölről kezdődik, ha nem, akkor a program végrehajtása a következő utasítással folytatódik.

Numerikus függvények

ABS(X) (ABSolute)

Abszolút érték. Az eredmény mindig pozitív.

ATN(X) (ArcustaNgens)

Árkusz tangens. Az eredményt radiánban kapjuk.

COS(X) (COSinus)

Koszinusz. Az X szöget radiánban kell megadni.

EXP(X) (EXPOnen)

Exponenciális függvény. Az eredmény e (2.71827183) X-edik hatványa.

FNxx(X)

A felhasználó által a DEFFN utasítással definiált függvény.

INT(X) (INTeger)

Egész rész. Levágja az X tizedesjegyeit. Az eredmény mindig kisebb vagy egyenlő, mint X. Ezért pl. INT(-2.1)=-3.

LOG(X) (LOGaritmus)

Természetes (e=2.71827183) alapú logaritmus. (10-es alapú logaritmus log(10)-zel való osztással válthatjuk át.)

PEEK(X)

Kiolvassa az X című tárcellában levő adatot, ahol X 0 és 65535 közötti egész szám lehet. Az eredmény 0 és 255 közötti egész. Gyakran a POKE utasítással együtt használatos.

RND(X) (RaNDom)

Visszatérési értéke egy 0 és 1 közé eső véletlenszám. Ha az első véletlenszámot RND(-TI)-vel generáljuk, akkor minden futtatásnál más és más kezdőértéket kapunk.

Egy negatív X paraméterrel új kezdőértéket állíthatunk be. Ugyanaz a negatív szám mindig ugyanazt a véletlenszám-sorozatot eredményezi, ha a sorozat következő elemeit RND(1)-gyel generáljuk. RND(0)-val mindig új sorozatot kapunk.

Egy adott tartományba eső egész véletlenszámot a következő képlettel állíthatunk elő:

```
N=INT(RND(1)*Y)+X
```

ahol X az alsó határ, Y pedig a kívánt tartomány hossza.

SGN(X) (SiGNum)

Előjel függvény. Visszatérési értéke:

SGN(X)= –1 ha $X < 0$
SGN(X)= 0 ha $X = 0$
SGN(X)= +1 ha $X > 0$

SIN(X) (SINus)

Szinusz. Az X szöget radiánban kell megadnunk.

SQR(X) (SQuaRe)

Négyzetgyök. X csak pozitív vagy 0 lehet. Negatív X-re az ILLEGAL QVANTITY (Illegális mennyiség) hibaüzenetet kapjuk.

TAN(X) (TANgens)

Tangens. Az X szöget radiánban kell megadnunk.

USR(X) (USeR)

Felhasználói függvény. Meghívásakor a program arra a gépi nyelvű alprogramra ugrik, amelynek kezdőcímét a 785–786-os című tárcellák tartalmazzák. Az X argumentummal a BASIC program paramétert adhat át, ill. fogadhat a gépi kódú rutintól.

Karakterfüzér függvények

ASC(X$) (ASCII)

X$ első karakterének ASCII kódját adja meg.

CHR$(X) (CHaRacter)

Az ASC függvény fordítottja, visszatérési értéke az a karakter, amelynek ASCII kódja X.

LEFT$(X$,X)

Az X$ bal oldali X db karakteréből álló részkarakterfüzért adja vissza.

LEN(X$)

Az X$-ben levő karakterek (beleértve a betűközöket is) számát adja meg.

MID$(X$,S,X)

Azt a részkarakterfüzért adja vissza, amely X karaktert tartalmaz X$-ből, az S-edik karakterrel kezdve.

RIGHT$(X$,X)

Az X$ jobb oldali X db karakteréből álló részkarakterfüzért adja vissza.

STR$(X) (STRing)

Azt a karakterfüzért adja vissza, amely X kinyomtatott formájával azonos.

VAL(X$) (VALue)

Numerikus értéket képez X$-ből, így lényegében az STR függvény fordítottja. Balról jobbra halad végig X$-en, amíg egy számmá nem konvertálható karaktert nem talál. Az így meghatározott számsorozatot numerikus értékké alakítja át.

```
10 X = VAL("123.456")      X = 123.456
10 X = VAL("12A13B")       X = 12
10 X = VAL("RIU017")       X = 0
10 X = VAL("-1.23.15.67")  X = 1.23
```

Egyéb függvények

FRE(X) (FREE(X)

X értékétől függetlenül a tár szabad byte-jainak számát adja meg.

POS(X) (POSition)

X értékétől függetlenül annak az oszlopnak a számát adja meg a képernyőn, ahol a következő PRINT utasítás írni kezd.

SPC(X) (SPaCe)

Egy PRINT utasításba írva X karakterpozíciót ugorhatunk vele a képernyőn.

TAB(X) (TABulator)

Egy PRINT utasításba érve a nyomtatás az X-edik oszlopban kezdődik. (A nyomtató ugyanúgy keresi, mint SPC(X)-et.)

D Függelék

BASIC kulcsszavak rövidítése

A programok beírásakor időt takaríthatunk meg, ha a leggyakrabban használt BASIC kulcsszavakat rövidítjük. A PRINT rövidítése pl. a (?) kérdőjel. A rövidített bevitel módja az, hogy a kulcsszó első vagy első két betűjének beírása után a következő betűt a SHIFT-tel együtt nyomjuk le. A LIST parancs kiadása után megjelenő programlistában a kulcsszavak már teljesen kiírt alakjukban szerepelnek.

Parancs	Rövidítés	Megjelenés
ABS	A SHIFT B	A
AND	A SHIFT N	A ╱
ASC	A SHIFT S	A ♥
ATN	A SHIFT T	A
CHR$	C SHIFT H	C
CLOSE	CL SHIFT O	CL
CLR	C SHIFT L	C
CMD	C SHIFT M	C ╲
CONT	C SHIFT O	C
DATA	D SHIFT A	D ♠
DEF	D SHIFT E	D
DIM	D SHIFT I	D
END	E SHIFT N	E ╱
EXP	E SHIFT X	E ♣
FOR	F SHIFT O	F
FRE	F SHIFT R	F
GET	G SHIFT E	G
GOSUB	GO SHIFT S	GO ♥
GOTO	G SHIFT O	G
INPUT#	I SHIFT N	I ╱
LET	L SHIFT E	L
LEFT$	LE SHIFT F	LE
LIST	L SHIFT I	L
LOAD	L SHIFT O	L
MID$	M SHIFT I	M
NEXT	N SHIFT E	N
NOT	N SHIFT O	N
OPEN	O SHIFT P	O
PEEK	P SHIFT E	P
POKE	P SHIFT O	P
PRINT	?	?
PRINT#	P SHIFT R	P
READ	R SHIFT E	R
RESTORE	RE SHIFT S	RE ♥
RETURN	RE SHIFT T	RE
RIGHT$	R SHIFT I	R
RND	R SHIFT N	R ╱
RUN	R SHIFT U	R
SAVE	S SHIFT A	S ♠
SGN	S SHIFT G	S
SIN	S SHIFT I	S
SPC(	S SHIFT P	S

Parancs	Rövidítés	Megjelenés
SQR	S SHIFT Q	S ■
STEP	ST SHIFT E	ST ▔
STOP	S SHIFT T	S ▏
STR$	ST SHIFT R	ST ▁
SYS	S SHIFT Y	S ▕
TAB	T SHIFT A	T ♠
THEN	T SHIFT H	T ▕
USR	U SHIFT S	U ♥
VAL	V SHIFT A	V ♠
VERIFY	V SHIFT E	V ▔
WAIT	W SHIFT A	W ♠

E FÜGGELÉK

Képernyőkódok

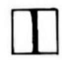
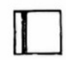
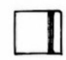
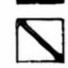
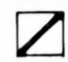
A következő táblázat a teljes COMMODORE-64 karakterkészletet tartalmazza. Azokat a kódokat foglalja össze, amelyeket a képernyőtár celláiba (1024–2023) kell írnunk (POKE) a kívánt karakterek megjelenítéséhez. Azt is megmutatja, hogy egy PEEK-kel kiolvasott értéknek melyik karakter felel meg.

Két különböző karakterkészlet áll rendelkezésünkre, de egyidőben mindig csak az egyiket használhatjuk. Ez azt jelenti, hogy nem lehet egyidejűleg két olyan karaktert megjeleníteni a képernyőn, amelyek közül az egyik az első, a másik pedig a második karakterkészlet eleme. A két készlet a SHIFT és C= billentyűk egyidejű lenyomásával cserélgethető. A BASIC-ben a POKE 53272,21 utasítás a nagybetű üzemmódot állítja be.

Ha a táblázatban megadott kódhoz 128-at adunk, akkor az illető karakter inverz formában (világos alapon sötét) jelenik meg.

A képernyő(tár) minden cellájához hozzá tartozik a színtár (55296–56295) egy cellája, amelynek tartalma meghatározza az adott helyre írt karakter színét. A G Függelék tartalmazza a képernyőtár és színtár térképét a színkódokkal együtt.

Képernyőkódok

1. Készlet	2. Készlet	Poke
@		0
A	a	1
B	b	2
C	c	3
D	d	4
E	e	5
F	f	6
G	g	7
H	h	8
I	i	9
J	j	10
K	k	11
L	l	12
M	m	13
N	n	14
O	o	15
P	p	16
Q	q	17
R	r	18
S	s	19
T	t	20
U	u	21
V	v	22
W	w	23
X	x	24
Y	y	25
Z	z	26
[		27
£		28
]		29
↑		30
←		31
SPACE		32
!		33
"		34
#		35
S		36
%		37
&		38
'		39
(		40
)		41
*		42
+		43
,		44
–		45
.		46
/		47
0		48
1		49
2		50
3		51
4		52
5		53
6		54
7		55
8		56
9		57
:		58
;		59
<		60
=		61
>		62
?		63
		64
	A	65
	B	66
	C	67
	D	68
	E	69
	F	70
	G	71
	H	72
	I	73
	J	74
	K	75
	L	76
	M	77
	N	78
	O	79
	P	80

1. Készlet	2. Készlet	Poke
	Q	81
	R	82
	S	83
	T	84
	U	85
	V	86
	W	87
	X	88
	Y	89
	Z	90
		91
		92
		93
		94
		95
SPACE		96
		97
		98
		99
		100
		101
		102
		103
		104
		105
		106
		107
		108
		109
		110
		111
		112
		113
		114
		115
		116
		117
		118
		119
		120
		121
		122
		123
		124
		125
		126
		127

Megjegyzés: SPACE = betűköz

A 128–255 kódok a 0–127 kódú karakterek inverz formáját adják. A második készlet az üresen hagyott helyeken megegyezik az elsővel.

F FÜGGELÉK

ASCII és CHR$ kódok

A következő összefoglaló a COMMODORE 64-essel megjeleníthető karakterek CHR$ kódjait tartalmazza. Ezt a táblázatot általában a CHR$( ) és az ASC( ) függvény argumentumának megállapításához használjuk. Nélkülözhetetlen azonban pl. a GET utasítással beolvasott karakterek kiértékeléséhez is.

A kinyomtatható karakterek kódjai mellett a funkciós kódokat is tartalmazza: képernyőtörlés, nagybetű/kisbetű átkapcsolás stb.

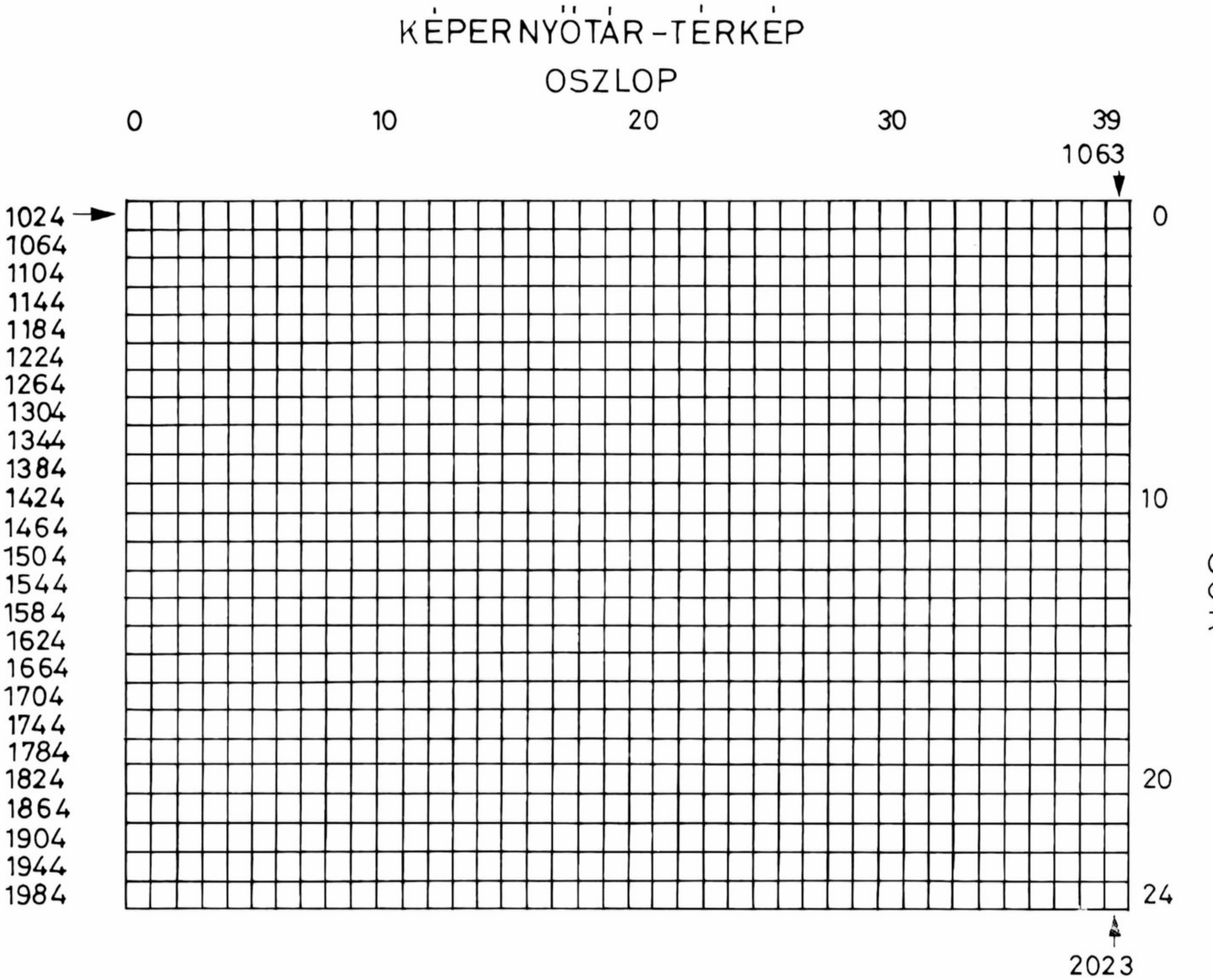
Karakter	CHR$	Karakter	CHR$	Karakter	CHR$	Karakter	CHR$
	0		23	.	46	E	69
	1		24	/	47	F	70
	2		25	0	48	G	71
	3		26	1	49	H	72
	4		27	2	50	I	73
WHI	5	RED	28	3	51	J	74
	6	CRSR →	29	4	52	K	75
	7	GRN	30	5	53	L	76
tilt SHIFT C=	8	BLU	31	6	54	M	77
engedélyez SHIFT C=	9	SPACE	32	7	55	N	78
	10	!	33	8	56	O	79
	11	"	34	9	57	P	80
	12	#	35	:	58	Q	81
RETURN	13	$	36	;	59	R	82
ÁTKAPCSOLÁS KISBETŰRE	14	%	37	<	60	S	83
	15	&	38	=	61	T	84
	16	'	39	>	62	U	85
CRSR	17	(	40	?	63	V	86
RVS ON	18	)	41	@	64	W	87
CLR HOME	19	*	42	A	65	X	88
INST DEL	20	+	43	B	66	Y	89
	21	,	44	C	67	Z	90
	22	–	45	D	68	[	91

Karakter	CHR$	Karakter	CHR$	Karakter	CHR$	Karakter	CHR$
£	92		124	PUR	156		184
]	93		125	← CRSR	157		185
↑	94		126	YEL	158		186
←	95		127	CYN	159		187
	96		128	SPACE	160		188
♠	97	narancs	129		161		189
	98		130		162		190
	99		131		163		191
	100		132		164		
	101	f 1	133		165		
	102	f 3	134		166		
	103	f 5	135		167		
	104	f 7	136		168		
	105	f 2	137		169		
	106	f 4	138		170		
	107	f 6	139		171		
	108	f 8	140		172		
	109	SHIFT RETURN	141		173		
	110	átkapcsol. nagybetüre	142		174		
	111		143		175		
	112	BLK	144		176		
	113	CRSR	145		177		
	114	RVS OFF	146		178		
♥	115	SHIFT CLR HOME	147		179		
	116	INST DEL	148		180		
	117	barna	149		181		
	118	vil.piros	150		182		
	119	szürke 1	151		183		
♣	120	szürke 2	152				
	121	vil.zöld	153				
♦	122	vil. kék	154				
	123	szürke 3	155				

192-233 kódok megegyeznek a 96-127 kódokkal 224-254 kódok megegyeznek a 160-190 kódokkal 255 kód megegyezik a 126 kóddal

G FÜGGELÉK

A képernyő- és a színtár térképe

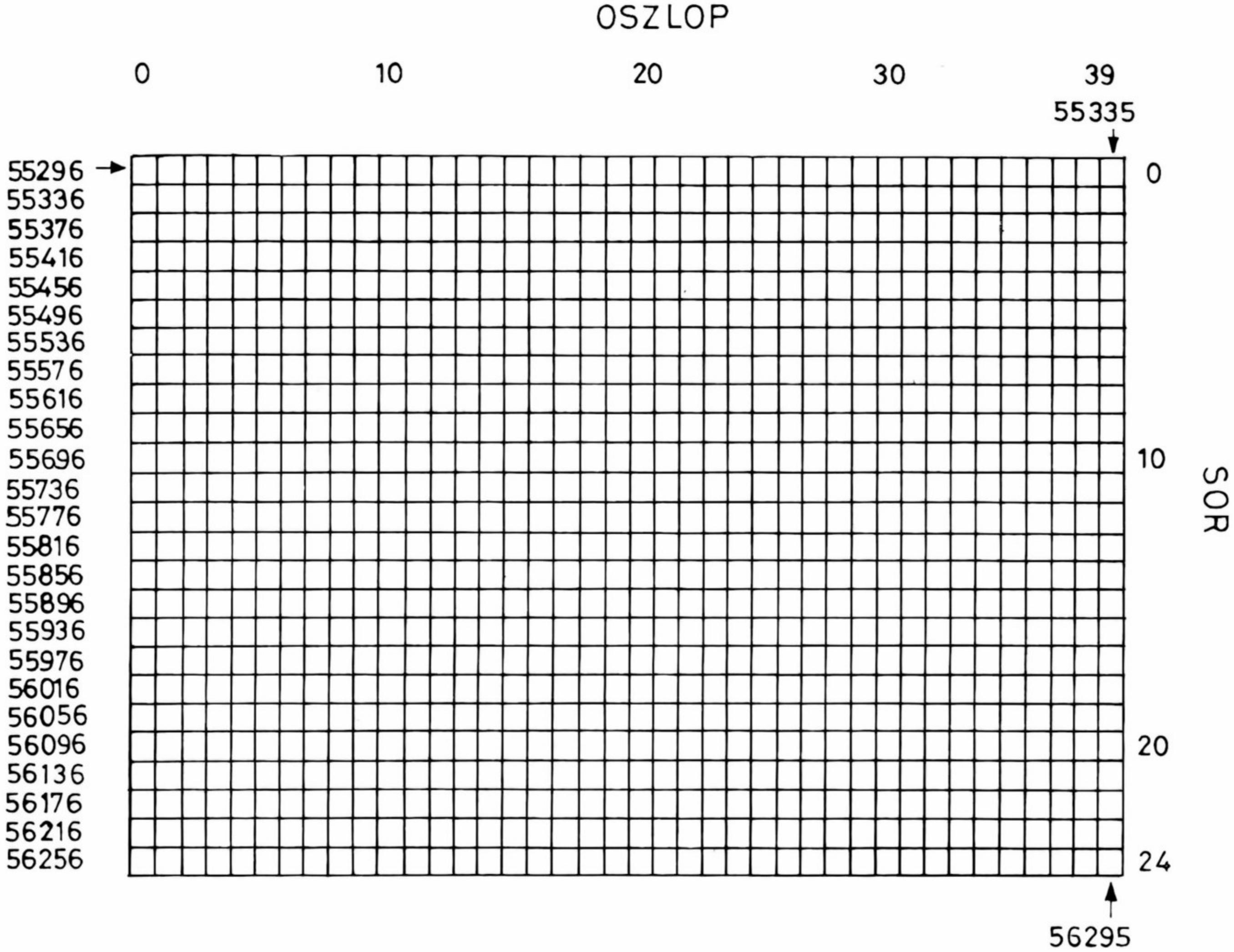


A következő táblázat a színek beállításához a színtárba írandó (POKE) értékeket foglalja össze:

0 fekete	8 narancs
1 fehér	9 barna
2 piros	10 halványpiros
3 türkiz	11 sötétszürke
4 lila	12 középszürke
5 zöld	13 világoszöld
6 kék	14 világoskék
7 sárga	15 világosszürke

Ha pl. a képernyő bal felső sarkában megjelenő karakter színét pirosra szeretnénk változtatni, akkor a következő POKE utasításra van szükség: POKE 55296,2

SZÍNTÁR-TÉRKÉP



H FÜGGELÉK

Származtatott matematikai függvények

A COMMODORE 64-es BASIC-ben előre nem definiált függvényei a következő összefüggésekkel számíthatók ki:

FÜGGVÉNY	BASIC EKVIVALENS
SZEKÁNS	SEC(X)=1/COS(X)
KOSZEKÁNS	CSC(X)=1/SIN(X)
KOTANGENS	COT(X)=1/TAN(X)
ÁRKUSZ SZINUSZ	ARSIN(X)=ATN(X/SQR(1-X↑2))
ÁRKUSZ KOSZINUSZ	ARCOS(X)=-ATN(X/SQR(1-X↑2))+π/2
ÁRKUSZ KOTANGENS	ARCOT(X)=ATN(X)+π/2
ÁRKUSZ SZEKÁNS	ARSEC(X)=ATN(X/SQR(X↑2-1))
ÁRKUSZ KOSZEKÁNS	ARCSC(X)=ATN(X/SQR(X↑2-1)) + (SGN(X)-1*π/2
SZINUSZ HIPERBOLIKUS	SINH(X)=(EXP(X)-EXP(-X))/2
KOSZINUS HIPERBOLIKUS	COSH(X)=(EXP(X)+EXP(-X))/2
TANGENS HIPERBOLIKUS	TANH(X)=EXP(-X)/(EXP(X) +EXP(-X))*2+1
KOTANGENS HIPERBOLIKUS	COTH(X)=EXP(-X)/(EXP(X) -EXP(-X))*2+1
SZEKÁNS HIPERBOLIKUS	SECH(X)=2/(EXP(X)+EXP(-X))
KOSZEKÁNS HIPERBOLIKUS	CSCH(X)=2/(EXP(X)-EXP(-X))
ÁRKUSZ SZINUSZ HIPERBOLIKUS	ARSINH(X)=LOG(X+SQR(X↑2+1))
ÁRKUSZ KOSZINUSZ HIPERBOLIKUS	ARCOSH(X)=LOG(X+SQR(X↑2-1))
ÁRKUSZ TANGENS HIPERBOLIKUS	ARTANH(X)=LOG((1+X)/(1-X))/2
ÁRKUSZ KOTANGENS HIPERBOLIKUS	ARCOTH(X)=LOG((X+1)/(X-1))/2
ÁRKUSZ SZEKÁNS HIPERBOLIKUS	ARSECH(X)=LOG((SQR(1-X↑2)+1)/X)
ÁRKUSZ KOSZEKÁNS HIPERBOLIKUS	ARCSCH(X)=LOG((SQR(1+X↑2)+1/X) *SGN(X)

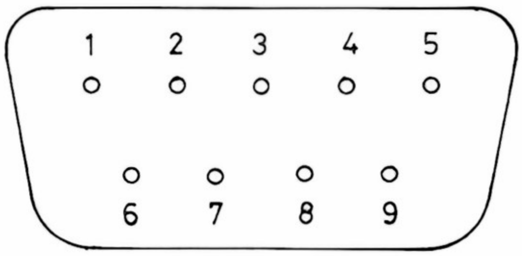
I FÜGGELÉK

Csatlakozókiosztás

1) Játékcsatlakozók
2) Modulcsatlakozó
3) Audio/Video
4) Soros B/K (Lemez/Nyomtató)
5) Kazetta
6) Felhasználói port
*) Button= a botkormány nyomógombja
   LP= Fényceruza
**) POT=Paddle potenciométer

JÁTÉKCSATLAKOZÓ 1

Pólus	Jel	Megjegyzés
1	JOYA 0	
2	JOYA 1	
3	JOYA 2	
4	JOYA 3	
5	POT AY**	
6	BUTTON A/LP*	
7	+5V	MAX. 100 mA
8	GND	
9	POT AX**	



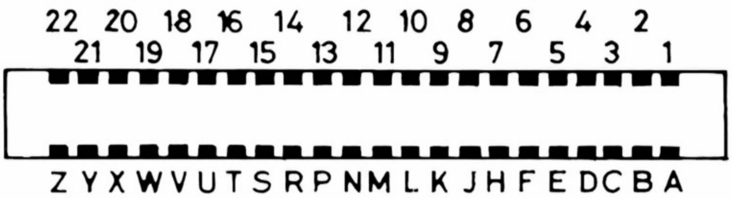
JÁTÉKCSATLAKOZÓ 2

Pólus	Jel	Megjegyzés
1	JOYB0	
2	JOYB1	
3	JOYB2	
4	JOYB3	
5	POT BY**	
6	BUTTON B	
7	+5V	MAX.100 mA
8	GND	
9	POT BX**	

MODULCSATLAKOZÓ

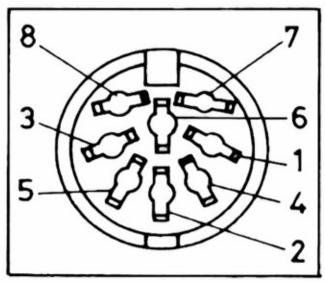
Pólus	Jel
22	GND
21	CD0
20	CD1
19	CD2
18	CD3
17	CD4
16	CD5
15	CD6
14	CD7
13	$\overline{\text{DMA}}$
12	$\overline{\text{BA}}$
11	$\overline{\text{ROML}}$
10	$\overline{\text{I/O2}}$
9	$\overline{\text{EXROM}}$
8	$\overline{\text{GAME}}$
7	$\overline{\text{I/O1}}$
6	Dot Clock
5	CR/$\overline{\text{W}}$
4	$\overline{\text{IRQ}}$
3	+5V
2	+5V
1	GND

Pólus	Jel
Z	GND
Y	CA0
X	CA1
W	CA2
V	CA3
U	CA4
T	CA5
S	CA6
R	CA7
P	CA8
N	CA9
M	CA10
L	CA11
K	CA12
J	CA13
H	CA14
F	CA15
E	$\phi 2$
D	$\overline{\text{NMI}}$
C	$\overline{\text{RESET}}$
B	$\overline{\text{ROMH}}$
A	GND



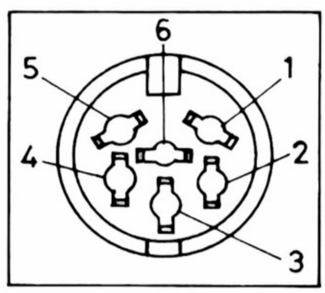
AUDIO/VIDEO

Pólus	Jel	Pólus	Jel
1	LUMINANCE	6	CROMINANZ
2	GND	7	O. BELEGUNG
3	AUDIO OUT	8	O. BELEGUNG
4	VIDEO OUT		
5	AUDIO IN		



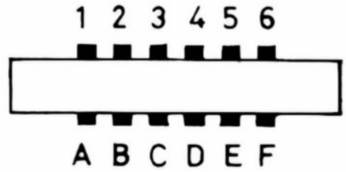
SOROS B/K

Pólus	Jel
1	SERIAL $\overline{\text{SRQIN}}$
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	$\overline{\text{RESET}}$



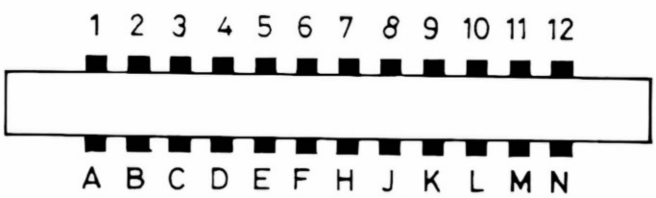
KAZETTA

Pólus	Jel
A-1	GND
B-2	+5V
C-3	CASETTE MOTOR
D-4	CASETTE READ
E-5	CASETTE WRITE
F-6	CASETTE SENSE



FELHASZNÁLÓI PORT

Pólus	Jel	Megjegyzés
1	GND	
2	+5V	MAX. 100 mA
3	$\overline{\text{RESET}}$	
4	CNT1	
5	SP1	
6	CNT 2	
7	SP2	
8	$\overline{\text{PC2}}$	
9	SER ATN IN	
10	9 VAC	MAX. 100 mA
11	9 VAC	MAX. 100 mA
12	GND	
A	GND	
B	$\overline{\text{FLAG}}$	
C	PB0	
D	PB1	
E	PB2	
F	PB3	
H	PB4	
J	PB5	
K	PB6	
L	PB7	
M	PA 2	
N	GND	



J FÜGGELÉK

Gyakorlóprogramok

A következő példaprogramok nemcsak hasznosak, de szórakoztatóak is. Próbáljuk ki őket!

```
100 PRINT "■■                    KODFEJTES"
110 INPUT"■LATNI AKARJA A SZABALYOKAT (I/N)"; Z$";
112 IF ASC(Z$)=78 THEN 170
120 PRINT"■TALALJA KI A TITKOS KODSZOT,"
130 PRINT"■■AMELY 5 BETUBOL ALL !"
140 PRINT "■MINDEN TALALGATAS UTAN MEGADOM"
150 PRINT "■A HELYES BETUK SZAMAT."
160 PRINT "■■AZT TANACSOLOM,"
165 PRINT "■HOGY NE VAKTABAN TALALGASSON !!!"
170 DATA JSPEB,JT[JL,KFMF[,LBSPN,MBQPT
180 DATA MBTTV,NFOFU,NJOUB,NPEPS,NVMBU
190 DATA NVOLB,NVUBU,OBQPT,QJIFO,QPDPL
200 DATA DJDVT,DVLPS,NBMBD,EBSBC,EPCPH
210 DATA EPMPH,EVSWB,FMFKU,GBLBE,GBSBH
220 DATA GFKFT,GFMBE,GJ[FU,GPEPS,GPHBE
230 DATA GPOBU,IBUPM,IFHFE,IFWFS,IJSFT
240 DATA BCMBL,BMCVN,BMLBU,BMLPU,IJUFM
250 DATA IPEJU,BNFMZ,CJCPS,CPLPS,DJGSB
260 DATA IPOBQ,IPNPL,JOHFS,JOEVM,JTNFS
270 N=50
280 DIM N$(N), Z(5), Y(5)
290 FOR J=1 TO N : READ N$(J) : NEXT J
300 T=TI
310 T=T/1000 : IF T>=1 THEN 310
320 Z=RND(-T)
330 G=0 : N$=N$(RND(1)*N+1)
340 PRINT "■■A KOD ERTELMES 5 BETUS SZO !"
380 G=G+1 : INPUT "■■MIT TIPPEL";Z$
390 IF LEN(Z$)<>5 THEN PRINT "■■5 BETUS SZOT KEREK !!!";
392 GOTO 380
400 V=0 : M=0 : H=0
410 FOR J=1 TO 5
420 Z=ASC(MID$(Z$,J,1)) : Y=ASC(MID$(N$,J,1))-1";
422 IF Y=64 THEN Y=90
430 IF Z<65 OR Z>90 THEN PRINT "■ERVENYTELEN KOD !";
432 GOTO 380
440 IF Z=65 OR Z=69 OR Z=73 OR Z=79 OR Z=85 OR Z=89";
442 THEN V=V+1
450 IF Z=Y THEN M=M+1
```

```
460 Z(J)=Z : Y(J)=Y : NEXT J
470 IF M=5 THEN 580
480 IF V=0 OR V=5 THEN PRINT "MI EZ A BUTASAG ???";
482 GOTO 380
490 FOR J=1 TO 5 : Y=Y(J)
500 FOR K=1 TO 5 : IF Y=Z(K) THEN H=H+1 : Z(K)=0";
502 GOTO 520
510 NEXT K
520 NEXT J
530 PRINT ""; H; "BETUT TALALT EL !"
540 IF G<30 THEN 380
550 PRINT "JOBB LESZ HA MEGMONDOM ! A KODSZO: ";
560 FOR J=1 TO 5 : PRINT CHR$(Y(J)); : NEXT J
570 GOTO 590
580 PRINT G "PROBALKOZASSAL TALALTA KI !"
590 INPUT "KER EGY UJ SZOT (I/N) "; Z$
600 R=1 : IF ASC(Z$)<>78 THEN 330

READY.
```

```
100 REM KOMPONALAS
110 SI=54272 : W1=SI+4 : W2=SI+11 : W3=SI+18
120 FO(0)=3000 : FO(1)=1508 : FO(2)=6030 .
130 DO=180
140 HT=2↑(1/12) : REM FELHANGKOZ
150 DEF FNH(F)=INT(F/256) : REM HI-BYTE
160 DEF FNL(F)=F-256*FNH(F) : REM LO-BYTE
170 FOR K=0 TO 20 : READ X : POKE SI+K,X : NEXT
172 REM HANGBEALLITAS
180 READ WA, WB, WC : REM HULLAMFORMAK
190 DIM F(100,6)
200 PRINT" SZOLAM-1  SZOLAM-2  SZOLAM-3  HOSSZ"
210 N=N+1 : FOR K=0 TO 2 : READ F : REM HANG AZ ADATOKBOL
220 F=INT(FO(K)*HT↑F+.5) : HB=FNH(F) : LB=FNL(F)
222 PRINT HB; LB, : REM HI ES LO
230 F(N,2*K)=LB : F(N,2*K+1)=HB
240 NEXT K
250 READ D : PRINT "  "D : F(N,6)=D : REM HOSSZ
260 IF D>0 THEN 210
270 POKE SI+24,15
280 FOR I=1 TO N-1
290 FOR K=0 TO 2 : POKE SI+7*K,F(I,2*K)
292 POKE SI+7*K+1,F(I,2*K+1) : NEXT K
300 POKE W1,WA : POKE W2,WB : POKE W3,WC
302 REM GENERATOR BEKAPCSOLAS
310 FOR K=1 TO F(I,6)*DO-150 : NEXT
320 POKE W1,0 : POKE W2,0 : POKE W3,0
322 REM GENERATOR KIKAPCSOLAS
330 FOR K=1 TO 50 : NEXT
340 NEXT I
350 GOTO 280
```

```
360 DATA 0,0,0,1,0,0,255 : REM SID REGISZTER
370 DATA 0,0,0,1,0,0,255
380 DATA 0,0,0,1,0,0,255
390 DATA 37,33,33 : REM HULLAMFORMA
400 DATA 0,0,0,2 : REM DALLAM
410 DATA 4,0,0,2
420 DATA 7,4,0,3
430 DATA 4,0,0,1
440 DATA 7,4,0,2
450 DATA 9,5,0,2
460 DATA 7,4,-5,3
470 DATA -99,-99,-99,1
480 DATA 4,0,0,2
490 DATA 7,0,0,2
500 DATA 9,5,5,8
510 DATA 7,4,0,3
520 DATA -99,-99,-99,1
530 DATA 4,0,0,2
540 DATA 7,4,0,2
550 DATA 7,4,0,3
560 DATA 4,0,0,1
570 DATA 5,2,0,2
580 DATA 4,0,0,2
590 DATA 2,-1,-5,3
600 DATA -99,-99,-99,1
610 DATA 0,-5,0,2
620 DATA 2,-1,0,2
630 DATA 4,0,0,4
640 DATA 2,-5,-5,4
650 DATA 0,0,0,3
660 DATA -99,-99,-99,1
670 DATA -99,-99,-99,-1

READY.
```

```
100 REM ABC
110 DIM A$(26)
120 Z$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
130 Z1$="12345678901234567890123456"
140 PRINT "■A JATAK CELJA A BETUK"
150 PRINT "■ABC SORRENDBE RENDEZESE"
160 PRINT "■MINDIG A BETUSOROZAT ELEJE"
170 PRINT "■FORGATHATO MEG. SOK SIKERT !"
180 PRINT "■■MILYEN HOSSZU LEGYEN A SOROZAT■"
190 INPUT "(A MAXIMALIS HOSSZ 26)  "; S
200 IF S<1 OR S>26 THEN 180
210 FOR I=1 TO S : A$(I)=MID$(Z$,I,1) : NEXT I
220 REM OSSZEKEVERES
230 FOR I=1 TO S : K=INT(RND(1)*S+1)
240 T$=A$(I) : A$(I)=A$(K) : A$(K)=T$
250 NEXT I
```

```
260 GOSUB 470 : T=0
270 REM RESZFUZER FORDITAS
280 T=T+1
290 INPUT "HANY BETUT FORDITSAK MEG"; R%
310 IF R%>0 AND R%<=S THEN 330
320 PRINT "▮1 ES"S"KOZE KELL ESNIE !▮" : GOTO 290
330 R=INT(R%/2) : FOR I=1 TO R
340 T$=A$(I)
350 T$=A$(I) : A$(I)=A$(R%-I+1) : A$(R%-I+1)=T$
360 NEXT I : GOSUB 470
370 REM SORREND VIZSGALAT
380 C=1 : FOR I=2 TO S
390 IF A$(I)>A$(I-1) THEN 410
400 C=0
410 NEXT I
420 IF C=0 THEN 270
430 PRINT "▮"T"▮PROBALKOZASSAL OLDOTTA MEG !"
440 INPUT "▮MEG EGYET"; Y$
450 IF LEFT$(Y$,1)="I" THEN 180
460 END
470 PRINT : PRINT LEFT$(Z1$,S)
480 FOR I=1 TO S : PRINT A$(I); : NEXT I
490 PRINT "▮" : RETURN

READY.
```

```
100 REM ELEKTRONIKUS ORGONA
110 PRINT "▮ ▮ ▮▮ ▮▮ | ▮▮ ▮▮ ▮▮ | ▮▮ ▮▮ | ▮▮ ▮▮ ▮▮ | "
120 PRINT "  ▮ ▮▮ ▮▮ | ▮▮ ▮▮ ▮▮ | ▮▮ ▮▮ | ▮▮ ▮▮ ▮▮ | "
130 PRINT "  ▮ ▮▮ ▮▮ | ▮▮ ▮▮ ▮▮ | ▮▮ ▮▮ | ▮▮ ▮▮ ▮▮ | "
140 PRINT "  ▮ | | | | | | | | | | | | | | |"
150 PRINT "  ▮Q |W |E |R |T |Y |U |I |O |P |@ |* |↑ |; |= |"
160 PRINT "▮        BETUKOZ :       PEDAL"
170 PRINT "▮'F1','F3','F5','F7': OKTAV"
180 PRINT "▮'F2','F4','F6','F8': HULLAMFORMA"
190 PRINT "▮VARJON EGY PICIT, FELKESZULOK !"
200 SI=13*4096+1024 : DIM F(26) : DIM K(255)
210 FOR I=0 TO 28 : POKE SI+I,0 : NEXT
220 F1=7939 : FOR I=1 TO 26 : F(27-I)=F1*5.8+30
222 F1=F1/2↑(1/12) : NEXT
230 K$="1Q2W3ER5T6Y7UI9O0P@-*▮↑▮;="
232 REM HASZNALT BILLENTYUK
240 FOR I=1 TO LEN(K$) : K(ASC(MID$(K$,I)))=I
242 NEXT
250 PRINT "▮                                                        "
260 AN=0 : AB=0 : HA=15 : AU=9 : HH=HA*16+AU
262 AS=AN*16+AB : WF=16 : M=1 : OK=4
265 HB=256 : Z=0
270 FOR I=0 TO 2 : POKE SI+5+I*7,AS
272 POKE SI+6+I*7,HT
```

```
280 POKE SI+2+I*7,4000 AND 255
282 POKE SI+3+I*7,4000/256 : NEXT
290 POKE SI+24,15 : REM +16 +64 : POKE SI+23,7
292 REM SZURO
300 GET A$ : IF A$="" THEN 300
310 FR=F(K(ASC(A$)))/M : FL=SI+V*7 : W=FL+4
312 IF FR=Z THEN 420
320 POKE FL+6,Z
330 POKE FL+5,Z : REM BURKOLOGORBE KI
340 POKE W,8 : POKE W,0 : REM RESET
350 POKE FL,FR-HB*INT(FR/HB) : REM FREK-LO
360 POKE FL+1,FR/HB : REM FREK-HI
370 POKE FL+6,HH : REM TARTAS
380 POKE W,WF+1 : FOR I=1 TO 50*AN : NEXT
390 POKE W,WF : REM KI
400 IF P=1 THEN V=V+1 : IF V=3 THEN V=0
410 GOTO 300
415 REM FUNKCIOBILL. LEKERDEZES
420 IF A$="■" THEN M=1 : OK=4 : GOTO 300
430 IF A$="■" THEN M=2 : OK=3 : GOTO 300
440 IF A$="■" THEN M=4 : OK=2 : GOTO 300
450 IF A$="■" THEN M=8 : OK=1 : GOTO 300
460 IF A$="■" THEN WF=16 : GOTO 300
470 IF A$="■" THEN WF=32 : GOTO 300
480 IF A$="■" THEN WF=64 : GOTO 300
490 IF A$="■" THEN WF=128 : GOTO 300
500 IF A$=" " THEN P=1-P
502 HH=(HH AND NOT 2) OR (NOT HH AND 2) : GOTO300
510 IF A$="■" THEN 210
520 GOTO 300

READY.
```

K FÜGGELÉK

BASIC programok honosítása a COMMODORE 64-esen

Ha olyan BASIC programokat szeretnénk felhasználni, amelyeket erèdetileg nem a COMMODORE 64-esre írtak, akkor a futtatás előtt várhatóan néhány apróbb javítást kell végeznünk. Ehhez szeretnénk most néhány tancsot adni.

Karakterfüzérek

Távolítsunk el minden olyan utasítást, amely a karakterfüzérek hosszát határozza meg. Pl. a DIM A$(I,J) definíciót, amely egy I hosszúságú elemekből felépülő J elemű tömböt dimenzionál, a COMMODORE BASIC-ben A$(J)-re kell módosítaniuk.

Egyes BASIC verziók a karakterfüzérek öszekapcsolásához (konkatenáció) a (,) vesszőt vagy az (&) jelet használják.

A COMMODORE 64-esen a részkarakterfüzérek képzésére a MID$, a RIGHT$ és a LEFT$ függvények alkalmazhatók. Tanulmányozzuk át gondosan a következő példát:

Egyéb BASIC verziók	COMMODORE 64 BASIC
A$(I)=X$	A$=LEFT$(A$,I-1)+X$+MID$(A$,I+1)
A$(I,J)=X$	A$=LEFT$(A$,I-1)+X$+MID$(A$,J+1)

Többszörös értékadás

Egyes BASIC verziókban a B és C változókat egyidejűleg a

```
10 LET B=C=0
```

utasítással nullázhatjuk. A COMMODORE 64-es a második egyenlőségjelet logikai műveletnek tekintené, így a várt eredményt kapnánk. Ezért a többszörös értékadásokat szét kell tördelnünk:

```
10 B=0 : C=0
```

ːöbbszörös utasítások

A COMMODORE 64-es BASIC az ugyanazon sorba írt többszörös utasításokat (:) kettősponttal választja el egymástól. Más BASIC verzióban erre a célra a (\) ɜl is előfordulhat.

ɅAT függvény

A COMMODORE 64-es BASIC nem ismeri a MAT függvényt, ezért a mátrixműveɜteket alkalmasan felépített FOR...NEXT ciklusokkal kell végrehajtanunk.

L FÜGGELÉK

Hibaüzenetek

A függelék a COMMODORE 64-es hibaüzeneteink teljes listáját, és az okok leírását tartalmazza.

BAD DATA (Hibás adat) – Egy adatállományból karakterfüzért olvasunk be, noha a program numerikus adatot vár.

BAD SUBSCRIPT (Hibás index) – A program olyan tömbelemre próbál hivatkozni, amelynek indexe kívül esik a DIM utasításban definiált határon.

CAN'T CONTINUE (Nem folytatható) – A CONT parancs nem hajtható végre, mert a program eddig még nem futott, hibás volt, vagy mert módosítottunk egy sort.

DEVICE NOT PRESENT (Nem jelenlevő készülék) – A szükséges bemeneti/kimeneti készülék nem áll az OPEN, CLOSE, CMD, PRINT#, INPUT# vagy GET# rendelkezésére.

DIVISION BY ZERO (Osztás nullával) – A nullával való osztás nem megengedett.

EXTRA IGNORED (Felesleges adatok eldobva) – Túl sok adatot vittünk be egy INPUT utasítás után. A gép a felesleget nem vette figyelembe.

FILE NOT FOUND (Nincs meg az állomány) – Az állomány keresése közben a szalagon elértük az END–OF–TAPE (szalagvég) jelet, ill. a lemezen nem létezik ilyen nevű adatállomány.

FILE NOT OPEN (Az állomány nincs nyitva) – A CMD, PRINT#, INPUT# vagy GET# parancsban hivatkozott állomány nincs megnyitva.

FILE OPEN (Az állomány nyitva van) – Olyan logikai állományszámot használunk egy állomány megnyitásakor, amely egy már megnyitott állományhoz tartozik.

FORMULA TOO COMPLEX (Túlságosan összetett kifejezés) – A karakterfüzért legalább két részre kell bontani, hogy a gép dolgozni tudjon vele.

ILLEGAL DIRECT (Illegális parancs) – Az INPUT utasítás csak programon belül használható, parancs üzemmódban nem.

ILLEGAL QUANTITY (Illegális mennyiség) – Az utasítás vagy függvény argumentumaként használt mennyiség kívül esik a megengedett tartományon.

LOAD (Betöltés) – A szalagon levő programmal valami probléma van.

NEXT WITHOUT FOR (FOR nélküli NEXT) – Hibásan ágyaztuk egymásba a ciklusokat, vagy a FOR-ban megadott változónév nem egyezik meg a NEXT-ben használttal.

NOT INPUT FILE (Nincs bemeneti állomány) – Olyan állományból próbáltunk adatokat olvasni INPUT#-tal vagy GET#-tel, amelyet csak kivitelre specifikáltunk.

NOT OUTPUT FILE (Nincs kimeneti állomány) – Olyan állományba próbálunk adatokat írni PRINT#-tel, amelyet csak olvasásra nyitottunk meg.

OUT OF DATA (Vége az adatoknak) – READ utasítást hajtunk végre, noha a DATA sorokból elfogytak már az adatok.

OUT OF MEMORY (Megtelt a tár) – Nincs már elég RAM-terület a program és a változók tárolásához. Ez a hiba léphet fel akkor is, ha túl sok az egymásba ágyazott alprogram vagy FOR...NEXT ciklus, ill. a zárójel.

OVERFLOW (Túlcsordulás) – A művelet eredménye nagyobb, mint a megengedett legnagyobb szám (1.70141183 E+38)

REDIM'D ARRAY (újradimenzionált tömb) – Egy tömböt csak egyszer dimenzionálhatunk. Ha egy tömbváltozóra a tömb dimenzionálása előtt hivatkozunk, akkor a gép automatikusan 10-re állítja az illető tömb dimenzióját. Minden ezután következő DIM utasítás ezt a hibajelzést váltja ki.

REDO FROM START (Bementi adathiba) – Egy INPUT utasítás után karakterfüzért vittünk be a várt numerikus adat helyett. A helyes érték begépelése utána a programvégrehajtás automatikusan folytatódik.

RETURN WITHOUT GOSUB (GOSUB nélküli RETURN) – A PROGRAM egy RETURN utasításra fut rá, noha előtte nem hajtat végre GOSUB-ot.

STRING TOO LONG (Túl hosszú karakterfüzér) – Egy karakterfüzér legfeljebb 255 karakterből állhat.

SYNTAX (Szintaxis hiba) – A COMMODORE 64-es nem ismer fel egy utasítást. Felesleges vagy hiányzó zárójelek, tévesen írt kulcsszavak stb. okozhatják.

TYPE MISMATCH (Nem egyező típus) – Karakterfüzér helyett számot használtunk vagy fordítva.

UNDEF'D FUNKTION (Definiálatlan függvény) – Olyan függvényre hivatkoztunk, amelyet még nem definiáltunk a DEF FN utasítással, ill. a definíciót tartalmazó sorra még nem futott rá a program.

UNDEF'D STATEMENT (Definiálatlan utasítás) – A GOTO, GOSUB vagy RUN utasítás nem létező sorszámra hivatkozik.

VERIFY (Ellenőrzés) – A szalagon vagy lemezen levő program nem egyezik meg a tárban levővel.

M FÜGGELÉK

Irodalom

Angerhausen–Brückmann–Englisch–Gerits: A Commodore 64-es belső felépítése. Novotrade Rt., Budapest, 1985.

Áts L.: Oxford Pascal C 64-esen. Novotrade Rt., Budapest, 1987.

Áts L.: Superbase 64. Novotrade Rt., Budapest, 1988.

Bartel–Kraas–Schrüfer: Számítógép és sakk. Novotrade Rt., Budapest, 1987.

Brückmann: A Commodore 64-es csatlakozási lehetőségei. Novotrade Rt., Budapest, 1988.

Dachsel: Zenekönyv a Commodore 64-eshez. Novotrade Rt., Budapest, 1986.

Heift: CAD – Bevezetés a számítógéppel segített műszaki tervezésbe. Novotrade Rt., Budapest, 1987.

Hornig–Trapp–Weltner: További tippek és trükkök a Commodore 64-esen. Novotrade Rt., Budapest, 1986.

Kampow: BASIC gyakorlatok a Commodore 64-esen. Novotrade Rt., Budapest, 1986.

Kerkloh–Tornsdorf: GEOS mindenkinek a Commodore 64-esre. Novotrade Rt., Budapest, 1988.

Liesert: PEEK-ek és POKE-ok a C 64-esen. Novotrade Rt., Budapest, 1987.

Pál Zs.–Révbíró T.: Hetedhét Commodore 64. Novotrade Rt., Budapest, 1985.

Plenge–Szczepanowski: Simon's BASIC gyakorlatok. Novotrade Rt., Budapest, 1986.

Sasse: Compiler (Commodore 64). Novotrade Rt., Budapest, 1987.

Severin: Tudomány és technika – Commodore 64. Novotrade Rt., Budapest, 1986.

Steigers: A robotok és a Commodore 64. Novotrade Rt., Budapest, 1986

Vadnai Sz.: C 64-es programozói zsebkönyv. Novotrade Rt., Budapest, 1986.

Voss: Bevezetés a statisztikai számításokba C 64-esen. Novotrade Rt., Budapest, 1986.

N FÜGGELÉK

SPRITE regiszter kiosztás

VIC Báziscím=53248$_{Dec}$=D000$_{Hex}$

Regiszter Dec	Hex	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
0	0	S0X7	S0X6	S0X5	S0X4	S0X3	S0X2	S0X1	S0X0	SPRITE 0 X
1	1	S0Y7							S0Y0	SPRITE 0 Y
2	2	S1X7							S1X0	SPRITE 1 X
3	3	S1Y7							S1Y0	SPRITE 1 Y
4	4	S2X7							S2X0	SPRITE 2 X
5	5	S2Y7							S2Y0	SPRITE 2Y
6	6	S3X7							S3X0	SPRITE 3X
7	7	S3Y7							S3Y0	SPRITE 3Y
8	8	S4X7							S4X0	SPRITE 4 X
9	9	S4Y7							S4Y0	SPRITE 4 Y
10	A	S5X7							S5X0	SPRITE 5 X
11	B	S5Y7							S5Y0	SPRITE 5 Y
12	C	S6X7							S6X0	SPRITE 6 X
13	D	S6Y7							S6Y0	SPRITE 6Y
14	E	S7X7							S7X0	SPRITE 7X
15	F	S7Y7							S7Y0	SPRITE 7 Y
16	10	S7X8	S6X8	S5X8	S4X8	S3X8	S2X8	S1X8	S0X8	X koord. leg-magasabb bit
17	11	RC8	EC5	BSM	BLNK	RSEL	YSCL2	YSCL1	YSCL0	
18	12	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	RASZTER
19	13	LPX7							LPX0	FÉNYCERUZA X
20	14	LPY7							LPY0	FÉNYCERUZA Y
21	15	SE7							SE0	SPRITE ENGEDÉLYEZÉS BE/KI
22	16	N.C.	N.C.	RST	MCM	CSEL	XSCL2	XSCL1	XSCL0	
23	17	SEXY 7							SEXY0	SPRITE NAGYÍTÁS Y

Regiszter# Dec	Hex	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
24	18	VS13	VS12	VS11	CB13	CB12	CB11	CB10	N.C.	Képernyő-karakter tár
25	19	IRQ	N.C.	N.C.	N.C.	LPIRQ	ISSC	ISBC	RIRQ	Megszakítás kérések
26	1A	N.C	N.C	N.C	N.C	MLPI	MISSC	MISBC	MRIRQ	Megszakítás kérés maszkok
27	1B	BSP7							BSP0	Háttér Sprite PRIORITÁS
28	1C	SCM7							SCM0	MULTIKOLOR SPRITE KIVÁLASZTÁS
29	1D	SEXX7							SEXX0	SPRITE NAGYÍTÁS X
30	1E	SSC7							SSC0	Sprite-Sprite ÜTKÖZÉS
31	1F	SBC7							SBC0	Sprite-Háttér ÜTKÖZÉS
		SZÍNINFORMÁCIÓK								
32	20									Képkeret
33	21									Háttér 0
34	22									Háttér 1
35	23									Háttér 2
36	24									Háttér 3
37	25						Sprite			SMC0
38	26						Multikolor			SMC1
39	27									Szín Sprite 0
40	28									Szín Sprite 1
41	29									Szín Sprite 2
42	2A									Szín Sprite 3
43	2B									Szín Sprite 4
44	26									Szín Sprite 5
45	2D									Szín Sprite 6
46	2E									Szín Sprite 7

A színkódokat lásd a 6. függelékben.

A multikolor üzemmódban csak a 0...7 színkódok használhatók.

O FÜGGELÉK

COMMODORE 64-es hanggenerálás

A következő táblázat a hanggenerálás kulcssszámait tartalmazza. A vezérlés módja a COMMODORE 64-es BASIC-ben:

POKE Regiszter, Paraméter

Az osztott regiszterek esetében a paramétereket alkalmas módon összegezni kell; pl. közepes felfutás és közepes csillapodás esetén a 2-es szólamban:

```
POKE 54272+12, 5*16+7 vagy POKE 54284,87
```

bázíscím (54272) regiszter (12) felfutás (5*16) csillapodás (7)

A hanggenerálást mindig meg kell előznie a hangerő-beállításnak. A 24-es regiszterbe írt (POKE) 0 és 15 közötti érték mindhárom szólamra érvényes.

Vezérlőregiszterek

SID báziscím: 54272 Dec= D400 Hex

<table>
<tr><th colspan="3">REGISZTER
SZÓLAM 1 2 3</th><th colspan="5">TARTALOM</th></tr>
<tr><td>0</td><td>7</td><td>14</td><td colspan="5">FREKVENCIA LO-BYTE (0... 255)</td></tr>
<tr><td>1</td><td>8</td><td>15</td><td colspan="5">FREKVENCIA HI-BYTE (0... 255)</td></tr>
<tr><td>2</td><td>9</td><td>16</td><td colspan="5">KITÖLTÉSI TÉNYEZŐ LO-BYTE (0...255)</td></tr>
<tr><td>3</td><td>10</td><td>17</td><td colspan="5">KITÖLTÉSI TÉNYEZŐ HI-BYTE(0...15)(CSAK NÉGYSZÖG HULLÁMRA)</td></tr>
<tr><td>4</td><td>11</td><td>18</td><td>Hullámforma:</td><td>Zaj
129</td><td>Négyszög
65</td><td>Fűrészfog
33</td><td>Háromszög
17</td></tr>
<tr><td>5</td><td>12</td><td>19</td><td colspan="2">LEFUTÁS
0*16(kemény)....15*16(lágy)</td><td colspan="3">CSILLAPODÁS
0(kemény)....15*16(lágy)</td></tr>
<tr><td>6</td><td>13</td><td>20</td><td colspan="2">TARTÁS
0*16(néma)... 15*16(hangos)</td><td colspan="3">LECSENGÉS
0(gyors)...15(lassú)</td></tr>
<tr><td>24</td><td>24</td><td>24</td><td colspan="5">HANGERŐ: 0(néma)... 15(teljes hangerő)</td></tr>
</table>

Példa: C hang az 5. oktávban, háromszög hullámformával, 2-es szólam

```
SI=54272

POKE SI+24,15 : POKE SI+7,207 : POKE SI+8,34 : POKE SI+13,240
```

POKE SI+24,15	POKE SI+7,207	POKE SI+8,34	POKE SI+13,240
(hangerő)	(frekvencia) lo–byte	(frekvencia) hi–byte	(hangtartás)

Hanggenerátor-bekapcsolás: POKE SI+11,17
Hanggenerátor-kikapcsolás: POKE SI+11,0

További SID regiszterek

<table>
<tr><th>REGISZTER</th><th colspan="8">TARTALOM</th></tr>
<tr><td>21</td><td colspan="8">SZŰRŐ HATÁRFREKVENCIA LO BYTE (0...7)</td></tr>
<tr><td>22</td><td colspan="8">SZŰRŐ HATÁRFREKVENCIA HI BYTE (0...255)</td></tr>
<tr><td rowspan="2">23</td><td colspan="4" rowspan="2">REZONANCIA
0(nincs)........15*16(erős)</td><td colspan="4">SZŰRŐ BEKAPCSOLÁS</td></tr>
<tr><td>külső
8</td><td>SZ3
4</td><td>SZ2
2</td><td>SZ1
1</td></tr>
<tr><td rowspan="2">24</td><td colspan="4">SZŰRŐ TÍPUS</td><td colspan="4" rowspan="2">HANGERŐ
0(néma)....10(hangos)</td></tr>
<tr><td>SZ3
ki
128</td><td>Felül
áteresztő
64</td><td>Sáv
áteresztő
32</td><td>Alul
áteresztő
16</td></tr>
</table>

A SID 4 olyan regisztert is tartalmaz, amelyek nem a hanggeneráláshoz kapcsolódnak. Ezekből a COMMODORE 64-es központi processzora a következő adatokat tudja kiolvasni:

REGISZTER	TARTALOM
25	KERÉK X
26	KERÉK Y
27	OSZCILLATOR R3
28	BURKOLÓGÖRBE 3

A vezérlőbemenetre kapcsolt forgatógombok helyzetének beolvasásához a következő utasításokat használhatjuk:

```
SI=54272:X=PEEK(SI+25):Y=PEEK(SI+26)
```

Az X és Y változók értéke ezután egy-egy 0 és 255 közötti szám lesz, amely a vezérlőgombok állását adja meg.

A 27-es és 28-as regiszterből a 3-as szólam aktuális oszcillátor, ill. burkológörbe értékét olvshatjuk be. Az így kapott számok segítségével pl. véletlenszámokat is képezhetünk. Ha valamilyen alkalmas algoritmus szerint felhasználjuk ezeket a többi szólam paraméterezésénél, akkor különleges hanghatásokat érhetünk el.

A következő paraméterekkel különböző hangszereket utánozhatunk:

HANGSZER	HULLÁMFORMA	LEÜTÉS	TARTÁS	KITÖLTÉSI TÉNYEZŐ
Zongora	Négyszög 65	9	0	HI-0, LO-255
Furulya	Háromszög 17	96	0	
Csembaló	Fűrészfog 33	9	0	
Xilofon	Háromszög 17	9	0	
Orgona	Háromszög 17	0	240	
Harmonika	Háromszög 17	102	0	
Trombita	Fűrészfog 33	96	0	

Megjegyzés: A burkológörbét mindig a hullámforma előtt kell beállítani (POKE).

P FÜGGELÉK

A hangjegyek frekvenciaparaméterei

A következő táblázat a teljes hangskála frekvenciaértékeit, valamint a SID regiszterekbe irandó (POKE) paramétereket tartamazza:

Sors.	Hangj.–Okt.	Frekvencia	Paraméter	Hi–By.	Lo–By.
0	C–0	16.4	278	1	22
1	C#–0	17.3	295	1	39
2	D–0	18.4	313	1	57
3	D#–0	19.4	331	1	75
4	E–0	20.6	351	1	95
5	F–0	21.8	372	1	116
6	F#–0	23.1	394	1	138
7	G–0	24.5	417	1	161
8	G#–0	26.0	442	1	186
9	A–0	27.5	468	1	212
10	A#–0	29.1	496	1	240
11	H–0	30.9	526	2	14
12	C–1	32.7	557	2	45
13	C#–1	34.6	590	2	78
14	D–1	36.7	625	2	113
15	D#–1	38.9	662	2	150
16	E–1	41.2	702	2	190
17	F–1	43.7	743	2	231
18	F#–1	46.2	788	3	20
19	G–1	49.0	834	3	66
20	G#–1	51.9	884	3	116
21	A–1	55.0	937	3	169
22	A#–1	58.3	992	3	224
23	H–1	61.7	1051	4	27
24	C–2	65.4	1114	4	90
25	C#–2	69.3	1180	4	156
26	D–2	73.4	1250	4	226
27	D#–2	77.8	1325	5	45
28	E–2	82.4	1403	5	123
29	F–2	87.3	1487	5	207
30	F#–2	92.5	1575	6	39
31	G–2	98.0	1669	6	133
32	G#–2	103.8	1768	6	232
33	A–2	110.0	1873	7	81
34	A#–2	116.5	1985	7	193

Sors.	Hangj.-Okt.	Frekvencia	Paraméter	Hi-By.	Lo-By.
35	H-2	123.5	2103	8	55
36	C-3	130.8	2228	8	180
37	C#-3	138.6	2360	9	56
38	D-3	146.8	2500	9	196
39	D#-3	155.6	2649	10	89
40	E-3	164.8	2807	10	247
41	F-3	174.6	2974	11	158
42	F#-3	185.0	3150	12	78
43	G-3	196.0	3338	13	10
44	G#-3	207.7	3536	13	208
45	A-3	220.0	3746	14	162
46	A#-3	233.1	3969	15	129
47	H-3	246.9	4205	16	109
48	C-4	261.6	4455	17	103
49	C#-4	277.2	4720	18	112
50	D-4	293.7	5001	19	137
51	D#-4	311.1	5298	20	178
52	E-4	329.6	5613	21	237
53	F-4	349.2	5947	23	59
54	F#-4	370.0	6301	24	157
55	G-4	392.0	6676	26	20
56	G#-4	415.3	7072	27	160
57	A-4	440.0	7493	29	69
58	A#-4	466.2	7939	31	3
59	H-4	493.9	8411	32	219
60	C-5	523.3	8911	34	207
61	C#-5	554.4	9441	36	225
62	D-5	587.3	10002	39	18
63	D#-5	622.3	10597	41	101
64	E-5	659.3	11227	43	219
65	F-5	698.5	11894	46	118
66	F#-5	740.0	12602	49	58
67	G-5	784.0	13351	52	39
68	G#-5	830.6	14145	55	65
69	A-5	880.0	14986	58	138
70	A#-5	932.3	15877	62	5
71	H-5	987.8	16821	65	181
72	C-6	1046.5	17821	69	157
73	C#-6	1108.7	18881	73	193
74	D-6	1174.7	20004	78	36
75	D#-6	1244.5	21193	82	201
76	E-6	1318.5	22454	87	182
77	F-6	1396.9	23789	92	237

Sors.	Hangj.-Okt.	Frekvencia	Paraméter	Hi–By.	Lo–By.
78	F#–6	1480.0	25203	98	115
79	G–6	1568.0	26702	104	78
80	G#–6	1661.2	28290	110	130
81	A–6	1760.0	29972	117	20
82	A#–6	1864.7	31754	124	10
83	H–6	1975.5	33642	131	106
84	C–7	2093.0	35643	139	59
85	C#–7	2217.5	37762	147	130
86	D–7	2349.3	40008	156	72
87	D#–7	2489.0	42387	165	147
88	E–7	2637.0	44907	175	107
89	F–7	2793.8	47578	185	218
90	F#–7	2960.0	50407	196	231
91	G–7	3136.0	53404	208	156
92	G#–7	3322.4	56580	221	4
93	A–7	3520.0	59944	234	40
94	A#–7	3729.3	63508	248	20

A megadott értékek természetesen nem kötelező jellegűek. Ha több szólamban "zenélünk", akkor a második és a harmadik szólamot kismértékben (!) elhangolva (frekvencia Lo–Byte) telt hangzáshoz jutunk.

Q FÜGGELÉK

A COMMODORE 64-es tárkiosztása

	Hexadecimális	Decimális	Leírás
	0000	0	6510 Adat irányregiszter
	0001	1	6510 Kiviteli regiszter
	0002	2	Nem használt
	0003–0004	3–4	Lebegőpontos–fixpontos átszámítási vektor
	0005–0006	5–6	Fixpontos–lebegőpontos átszámítási vektor
	0007	7	Keresőkarakter
	0008	8	Idézőjel üzemmód flag
	0009	9	TAB oszlopszámláló
	000A	10	0=LOAD, 1=VERIFY
	000B	11	Beviteli puffer mutató/elemszám
	000C	12	Standard DIM flag
	000D	13	Típus: FF=karakterfüzér 00= numerikus
	000E	14	Típus: 80= egész 00= lebegőpontos
	000F	15	DATA/LIST flag
	0010	16	Elem/FNx flag
	0011	17	00=INPUT, 40=GET, 98=READ
	0012	18	ATN előjel Egyenlőségi flag összehasonlításánál
	0013	19	aktuális B/K készülék
*	0014–0015	20–21	Egész érték
	0016	22	Átmeneti karakterfüzér-mutató
	0017–0018	23–24	Utolsó átmeneti karakterfüzér
	0019–0021	25–33	Átmeneti karakterfüzér
	0022–0025	34–37	Segédmutató terület
	0026–002A	38–42	Szorzat terület szorzásánál
*	002B–002C	43–44	BASIC kezdőmutató
*	002D–002E	45–46	Változó kezdőmutató
*	002F–0030	47–48	Tömb kezdőmutató
*	0031–0032	49–50	Tömb végmutató
*	0033–0034	51–52	Mutató a karakterfüzér területére (lefelé mozog)

(*=hasznos címek)

	Hexadecimális	Decimális	Leírás
	0035–0036	53–54	Karakterfüzér segédmutató
*	0037–0038	55–56	Mutató a tárhatárra
	0039–003A	57–58	Aktuális BASIC sor száma
	003B–003C	59–60	Előző BASIC sor száma
	003D–003E	61–62	BASIC utasítás mutató a CON-hoz
	003F–0040	63–64	Aktuális DATA sor száma
	0041–0042	65–66	Aktuális DATA elem címe
*	0043–0044	67–68	Ugrási vektor az INPUT-hoz
	0045–0046	69–70	Aktuális változónév
	0047–0048	71–72	Aktuális változó címe
	0049–004A	73–74	Változómutató a FOR...NEXT-hez
	004B–004C	75–76	Köztes tároló a BASIC mutatóhoz
	004D	77	Akkumulátor a karakter összehasonlításához
	004E–0053	78–83	Általános célú munkaterület (mutatók stb.)
	0054–0056	84–86	Ugrási vektor a függvényekhez
	0057–0060	87–96	Munkaterület a numerikus műveletekhez
*	0061	97	Lebegőpontos akkumulátor #1 exponens
*	0062–0065	98–101	Lebegőpontos akkumulátor #1 mantissza
*	0066	102	Lebegőpontos akkumulátor #1 előjel
	0067	103	Mutató a polinom kiértékeléshez
	0069–006E	105–110	Lebegőpontos akkumulátor #2 exponens stb.
	006F	111	Előjel összehasonlítás: Akku #1 / Akku #2
	0070	112	Akku #1 alsó helyi érték (kerekítés)
	0071–0072	113–114	Szalagpuffer hossza
*	0073–008A	115–138	CHRGET szubrutin
	007A–007B	122–123	BASIC mutató a szubrutinban
	008B–008F	139–143	Véletlenszám-generátor kezdőértéke
*	0090	144	ST státuszbyte
	0091	145	STOP és RVS billentyű flag
	0092	146	Időkonstans a szalaghoz
	0093	147	0=LOAD, 1=VERIFY
	0094	148	Soros kimenet: visszaadott karakter flag
	0095	149	Visszaadott karakter
	0096	150	EOT (szalagvég) érkezett
	0097	151	Regisztertár
*	0098	152	Nyitott állományok száma
*	0099	153	Beviteli készülék (normál=0)
*	009A	154	Kiviteli (CMD) készülék (normál=3)
	009B	155	Szalag paritásbyte

	Hexadecimális	Decimális	Leírás
	009C	156	Byte flag
	009D	157	Kivitelvezérlés (80=direkt, 00=RUN)
	009E	158	Hiba a szalag/karakterpufferben
	009F	159	Szalaghiba kijavítva
*	00A0–00A2	160–162	Belső óra
	00A3	163	Soros bitszámláló EOI/flag
	00A4	164	Ciklusszámláló
	00A5	165	Számláló (lefelé) a szalag írásánál
	00A6	166	Mutató a szalagpufferre
	00A7–00AB	167–171	Szalag olvasási és írási flagek
	00AC–00AD	172–173	Program startmutató
	00AE–00AF	174–175	Program végmutató
	00B0–00B1	176–177	Szalag időállandók
*	00B2–00B3	178–179	Mutató a szalagpuffer elejére
	00B4	180	Szalag időzítő, bitszámláló
	00B5	181	Szalag EOT/RS 232 következő bit adásra
	00B6	182	***
*	00B7	183	Karakterszám az állománynévben
*	00B8	184	Aktuális logikai állományszám
*	00B9	185	Aktuális másodlagos cím
*	00BA	186	Aktuális készülék
*	00BB–00BC	187–188	Mutató az állománynévre
	00BD	189	***
	00BE	190	Második blokkok száma (írás/olvasás)
	00BF	191	Soros szópuffer
	00C0	192	Kazettamotor flag
	00C1–00C2	193–194	B/K kezdőcím
	00C3–00C4	195–196	Vektorcím mutató, KERNAL
*	00C6	198	Karakterszám a billentyűzetpufferben
*	00C7	199	Képernyő RVS flag
	00C8	200	Sorvég mutató (bevitel)
	00C9–00CA	201–202	Beviteli kurzor helyzete (sor,oszlop)
*	00CB	203	Lenyomott billentyű (64=nincs)
	00CC	204	Kurzor be/ki (0=villogó kurzor)
	00CD	205	Villogó kurzor számláló
	00CE	206	Karakter a kurzorpozícióban
	00CF	207	Kurzor villogó állapotban
	00D0	208	Bevitel képernyőről, billentyűzetről
*	00D1–00D2	209–210	Képernyő sormutató
*	00D3	211	Képernyő oszlopmutató
	00D4	212	0=direkt kurzor, egyébként programozott
*	00D5	213	Aktuális képernyősor hossza (40/80)
*	00D6	214	Kurzort tartalmazó sor

	Hexadecimális	Decimális	Leírás
	00D7	215	Utolsó billentyű/ellenőrző összeg/puffer
*	00D8	216	Megengedett beszúrások száma
*	00D9–00F0	217–240	Képernyősor összekapcsoló táblázat
	00F1	241	Hamis képernyősor
	00F2	242	Képernyősor jelző
*	00F3–00F4	243–244	Mutató a képernyőszínre
	00F5–00F6	245–246	Mutató a billentyűzet dekódoló táblázatra
	00F7–00F8	247–248	RS 232 vételi mutató
	00F9–00FA	249–250	RS 232 átviteli mutató
*	00FB–00FE	251–254	Szabad terület a 0–ás lapon
	00FF	255	BASIC tár
	0100–010A	256–266	Lebegőpontos-ASCII átszámítás munkaterülete
	0100–013E	256–318	Szalaghiba
	0100–01FF	256–511	Processzor gyűjtőtár terület (heap)
*	0200–0258	512–600	BASIC beviteli puffer
*	0259–0262	601–610	Logika állomány táblázat
*	0263–026C	611–620	Készülékszám táblázat
*	026D–0276	621–630	Másodlagos cím táblázat
*	0277–0280	631–640	Billentyűzetpuffer
*	0281–0282	641–642	Rendszer RAM kezdőcím
*	0283–0284	642–644	Rendszer RAM végcím
	0285	645	Soros busz, időtúllépés flag
*	0286	646	Aktuális színkód
	0287	647	Szín a kurzor alatt
*	0288	648	Képernyőtár cím (lap)
*	0289	649	Billentyűzetpuffer maximális menete
*	028A	650	Billentyűismétlés (128=minden billentyű)
*	028B	651	Ismétlési sebesség számláló
	028C	652	Ismétlési késleltetés számláló
*	028D	653	SHIFT/CNTRL flag
	028E	654	Billentyűzet utolsó SHIFT mintája
	028F–0290	655–656	Mutató a billentyűzet dekódoló táblázatra
*	0291	657	SHIFT üzemmód (0=engedélyezés, 128=tiltás)
	0292	658	Automatikus úsztatás lefelé (0=be 0 =ki)
	0293	659	RS 232 Vezérlőregiszter
	0294	660	RS 232 Utasításregiszter
	0285–0296	661–662	nem Standard (bit idő) ****
	0297	663	RS 232 Státuszregiszter
	0298	664	Átviendő bitek száma
	0299–029A	665–666	Baudrate
	029B	667	RS 232 Vételi mutató

	Hexadecimális	Decimális	Leírás
	029C	668	RS 232 Bemeneti mutató vétel
	029D	669	RS 232 Átviteli mutató
	029E	670	RS 232 Kimeneti mutató adás
	029F–02A0	671–672	Kazettaműködés alatti IRQ vektor
	02A1–02FF	673–767	****
*	0300–0301	768–769	Hibaüzenet-vektor
	0302–0303	770–771	BASIC melegindítási vektor
	0304–0305	772–773	Kulcsszavak átalakítása zsetonokká
	0306–0307	774–775	Zsetonok átalakítása kulcsszavakká
	0308–0309	776–777	Új BASIC utasítás végrehajtása
	030A–030B	778–779	Aritmetika elem felhozása
	030C	780	6502 ”A regiszter
	030D	781	6502 ”X regiszter
	030E	782	6502 ”Y regiszter
	030F	783	6502 ”P regiszter
	0310–0313	784–787	USR ugrás
	0314–0315	788–789	Hardvermegszakítás (IRQ) (EA 31)
	0316–0317	790–791	Törésmegszakítás (FE66)
	0318–0319	792–793	Nem maszkolható megszakítás (NMI) (FE47)
	031A–031B	794–795	OPEN (F40A) (F34A)
	031C–031D	796–797	CLOSE (F291)
	031E–031F	798–799	Beviteli csatorna (F 2C7) (F 209)
	0320–0321	800–801	Kiviteli csatorna (F 250)
	0322–0323	802–803	B/K visszaállítás (az öszes nyitott csatorna törlése)
	0324–0325	804–805	INPUT(F 157)
	0326–0327	806–807	OUTPUT(F1CA)
	0328–0329	808–809	STOP billentyű vizsgálat (F770) (FGED)
	032A–032B	810–811	GET(F13E)
	032C–032D	812–813	Összes csatorna lezárva (F32F)
	032E–032F	814–815	Felhasználói IRQ(FE66)
	0330–0331	816–817	RAM töltés (F4A5)
	0332–0333	818–819	RAM tárolók (F5ED)
	0334–033B	820–827	***
	033C–03FB	828–1019	Kazettapuffer
	0400–07FF	1024–2047	1k képernyőtár
	(0400–07E7	1024–2023	Video mátrix)
	(07F8–07FF	2040–2047	SPRITE mutatók)
	0800–9FFF	2048–40959	BASIC felhasználói tár
	A000–BFFF	40960–49151	BASIC ROM
	C000–CFFF	49152–53247	4k RAM

R FÜGGELÉK

Képernyővezérlő karakterek

Az idézőjelbe foglalt karakterfüzérek beírásakor a COMMODORE 64-es egy különleges üzemmódba kerül. Ezt az ún. fenntartott üzemmódot mindig az idézőjelek bevitele kapcsolja be, ill. ki. Ilyenkor a billentyűzetről beadott képernyővezérlő parancsok (pl. kurzor jobbra) nem kerülnek végrehajtásra, hanem vezérlőkarakterként foglalnak helyet a beírt karakterfüzérben. A tényleges végrehajtást az váltja ki, hogy a programban egy PRINT utasítás meghívja az illető karakterfüzért.

A vezérlőjelek a képernyőn, ill. a nyomtatón speciális inverz karakter formájában jelennek meg. Ezt szemlélteti a következő programlista. A szimbólumok mögött az adott funkciót kiváltó billentyűt, ill. billentyűkombinációt adtuk meg.

```
100 REM VEZERLO KARAKTEREK A PRINT UTASITASBAN
110 PRINT "■ =SHIFT+CL       :KEPERNYOTORLES
120 PRINT "■ =HOME           :KURZOR HOME
130 PRINT "■ =CRSR←          :KURZOR A BAL FELSO SAROKBA
140 PRINT "■ =SHIFT+CRSR←    :KURZOR JOBBRA
150 PRINT "■ =CRSR↑          :KURZOR LEFELE
160 PRINT "■ =SHIFT+CRSR↑    :KURZOR FELFELE
165 PRINT "                  :SZINEK
170 PRINT "■ =CTRL+1         :FEKETE
180 PRINT "■ =CTRL+2         :FEHER
190 PRINT "■ =CTRL+3         :PIROD
200 PRINT "■ =CTRL+4         :TURKIZ
210 PRINT "■ =CTRL+5         :LILA
220 PRINT "■ =CTRL+6         :ZOLD
230 PRINT "■ =CTRL+7         :KEK
240 PRINT "■ =CTRL+8         .:SARGA
250 PRINT "■ =COMMO+1        :NARANCS
260 PRINT "■ =COMMO+2        :BARNA
270 PRINT "■ =COMMO+3        :HALVANYPIROS
280 PRINT "■ =COMMO+4        :SOTETSZURKE
290 PRINT "■ =COMMO+5        :KOZEPSZURKE
300 PRINT "■ =COMMO+6        :VILAGOSZOLD
310 PRINT "■ =COMMO+7        :VILAGOSKEK
320 PRINT "■ =COMMO+8        :VILAGOSSZURKE
330 PRINT "■ =CTRL+9         :INVERZ ABRAZOLAS BEKAPCSOLAS
340 PRINT "■ =CTRL+0         :INVERZ ABRAZOLAS KIKAPCSOLAS

READY.
```

S FÜGGELÉK

Nagyfelbontású grafika a COMMODORE 64-esen

Bár a C 64-es utasításkészlete nem tartalmaz grafikus parancsokat, a gép alkalmas grafikus jellegű felhasználásokra. A maximális felbontás 200x320 képpont. A képernyő aktuális állapotának (azaz a képernyőn megjelenített grafikának) a tárban egy bitminta felel meg. A 64000 képpont ábrázolásához szükséges bittérkép 8 kbyte helyett foglal le a tárban.

A nagyfelbontású grafikus alakzatok létrehozásának első lépése az 53272-es címen levő byte 3-as bitjének 1-re állítása (10-es sor), amivel a 8192-es cím feletti tárterületet a bittérképhez rendeljük. Ahhoz, hogy ez a bittérkép a képernyőn is megjelenjen, az 53265-ös byte 5-ös bitjét is 1-re kell billentenünk (20-as sor).

Most már csak azzal a problémával kell megküzdenünk, hogy milyen módon alakítsuk át bittérképpé a megjelenítendő alakzatot. A következő példaprogram egy szinuszfüggvény ábrázolásával szemléltet egy egyszerű módszert.

Ha a szinusz helyett egy tetszőleges függvényt szeretnénk felrajzolni, akkor a 60-as sort így kell módosítanunk:

```
Y=INT(f(X))
```

ahol X-re és Y-ra teljesülnie kell a következő feltételeknek:

```
0 < X < 319 és 0 < y < 199
```

Megjegyzés: Más színkombinációkat a 25-ös sorban a tárba írt értékek módosításával érhetünk el. Az alsó 4 bit a hátteret, a felső 4 bit a rajzolószínt határozza meg.

```
5 REM SINUSGORBE
10 POKE 53272,PEEK(53272) OR 8
12 REM BITTERKEP HOZZARENDELES
20 POKE 53265,PEEK(53265) OR 32
22 REM BITTERKEP UZEMMOD BEALLITAS
25 FOR K=0 TO 999 : POKE 1024+K,14 : NEXT
26 REM SZINKOMBINACIO KIVALASZTAS
27 FOR I=0 TO 7999 : POKE 8192+I,0 : NEXT
28 REM KEPERNYOTORLES
50 FOR X=0 TO 319
60 Y=INT(100+80*SIN(X*2*π/160)) : REM  FUGGVENY
70 FOR N=0 TO 24
80 IF Y>N*8-1 AND Y<(N+1)*8 THEN
82 BY=8192+N*320+8*(INT(X/8)+Y-8*N :N=24 :GOTO 100
85 REM BYTE SZAM MEGHATAROZAS
90 NEXT N
100 BI=8*(1+INT(X/8))-X-1 : REM BITMINTA KISZAMITAS
110 IF PEEK(BY)<>0 THEN POKE BY,PEEK(BY) OR 2↑BI
120 IF PEEK(BY)=0 THEN POKE BY,2↑BI
130 NEXT X
150 GOTO 150
```

T FÜGGELÉK

PRINT FRE(0)

A FRE(0) változót a gép egész számként tárolja, így értéke nem lehet nagyobb 32767-nél. Ha a szabad tárkapacitás (szabad byte-ok száma) ennél nagyobb, akkor a FRE(0) függvény negatív értéket ad. Ekkor a helyes eredményt úgy kapjuk meg, hogy a kapott értékhez 65535-öt hozzáadunk.

U FÜGGELÉK

A funkcióbillentyűk kódjai

Az F1–F8 funkcióbillentyűk a 133–140-es ASCII kódoknak felelnek meg (lásd ASCII kód lista, F Függelék). A következő rövid program a billentyűpufferek lekérdezését szemlélteti:

```
 1 REM FUNKCIO BILLENTYUK
10 GETA$:IFA$=""THEN 10
20 IFA$=CHR$(133)THEN 40
30 GOTO 10
40 PRINT"AZ F1 BILLENTYUT NYOMTA LE"
50 GOTO 10
```

V FÜGGELÉK

C 64-es programok betöltése CBM készüléken (3,4,8000)

A betöltés előtt írjuk be:

```
POKE 40,1 : POKE 41,8 : POKE 8*256,0 : NEW
```

A RETURN billentyű lenyomása után a program a szokásos módon betölthető és indítható.

Megjegyzés: A CBM programok betöltése a C 64-esbe minden további nélkül lehetséges, mivel a C 64-es a betöltési címet nem veszi figyelembe és minden programot a BASIC programtár elejétől kezdve (2048-as cím) helyez el. (Ez természetesen korlátozza BASIC 2.0-ban írt programokat.)

Z FÜGGELÉK

A kerék (paddle) és a botkormány (joystick) kezelése

Kerék:

Ha a kereket az 1-es PORT-ra kötjük, akkor a potenciométerek helyzetét jellemző értékeket az 54297-es és az 54298-as címről olvashatjuk be.

A tüzelőgombok megnyomását az 56321-es című byte 2-es és 3-as bitjének nullára álása (0) jelzi.

A 2-es PORT-ra kötött kerék esetében a tüzelőgombokhoz rendelt byte az 56320-as címen helyezkedik el. A potenciaméterek állását ugyanazok a címek tartalmazzák, mint az 1-es PORT esetében.

A 2-es PORT-ra az 56320-as címen levő byte 7-es bitjének 1-re billentésével kapcsolhatunk át. Mivel azonban ez a bit minden megszakításnál törlődik, a 2-es PORT-ra kapcsolt kereket csak egy gépi nyelvű programmal összhangban kérdezhetjük le.

Botkormány:

A botkormányok aktuális pozícióját az 56321-es (1-es PORT) és az 56320-as (2-es PORT) címről olvashatjuk be. A négy lehetséges állásnak (fel, le, balra, jobbra) sorban a 0-ás, 1-es, 2-es és 3-as bitek törlődése felel meg. A tüzelőgombok megnyomását mindkét PORT esetében a 4-es bit jelzi.